Network Working Group
Request for Comments: 1001 March, 1987

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS

ABSTRACT

This RFC defines a proposed standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC describes the NetBIOS-over-TCP protocols in a general manner, emphasizing the underlying ideas and techniques. Detailed specifications are found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications".

SUMMARY OF CONTENTS

TABLE OF CONTENTS

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS

1. STATUS OF THIS MEMO

This RFC specifies a proposed standard for the Internet community. Since this topic is new to the Internet community, discussions and suggestions are specifically requested.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Distribution of this document is unlimited.

2. ACKNOWLEDGEMENTS

This RFC has been developed under the auspices of the Internet Activities Board, especially the End-to-End Services Task Force.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

3. INTRODUCTION

This RFC describes the ideas and general methods used to provide NetBIOS on a TCP and UDP foundation. A companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications"[1] contains detailed descriptions of packet formats, protocols, and defined constants and variables.

The NetBIOS service has become the dominant mechanism for personal computer networking. NetBIOS provides a vendor independent interface for the IBM Personal Computer (PC) and compatible systems.

NetBIOS defines a software interface not a protocol. There is no "official" NetBIOS service standard. In practice, however, the IBM PC-Network version is used as a reference. That version is described in the IBM document 6322916, "Technical Reference PC Network"[2].

Protocols supporting NetBIOS services have been constructed on diverse protocol and hardware foundations. Even when the same foundation is used, different implementations may not be able to interoperate unless they use a common protocol. To allow NetBIOS interoperation in the Internet, this RFC defines a standard protocol to support NetBIOS services using TCP and UDP.

NetBIOS has generally been confined to personal computers to date. However, since larger computers are often well suited to run certain NetBIOS applications, such as file servers, this specification has been designed to allow an implementation to be built on virtually any type of system where the TCP/IP protocol suite is available.

This standard defines a set of protocols to support NetBIOS services.

These protocols are more than a simple communications service involving two entities. Rather, this note describes a distributed system in which many entities play a part even if they are not involved as an end-point of a particular NetBIOS connection.

This standard neither constrains nor determines how those services are presented to application programs.

Nevertheless, it is expected that on computers operating under the PC-DOS and MS-DOS operating systems that the existing NetBIOS interface will be preserved by implementors.

NOTE: Various symbolic values are used in this document. For their definitions, refer to the Detailed Specifications[1].

4. DESIGN PRINCIPLES

In order to develop the specification the following design principles were adopted to guide the effort. Most are typical to any protocol standardization effort; however, some have been assigned priorities that may be considered unusual.

4.1. PRESERVE NetBIOS SERVICES

In the absence of an "official" standard for NetBIOS services, the version found in the IBM PC Network Technical Reference[2] is used.

NetBIOS is the foundation of a large body of existing applications. It is desirable to operate these applications on TCP networks and to extend them beyond personal computers into larger hosts. To support these applications, NetBIOS on TCP must closely conform to the services offered by existing NetBIOS systems.

IBM PC-Network NetBIOS contains some implementation specific characteristics. This standard does not attempt to completely preserve these. It is certain that some existing software requires these characteristics and will fail to operate correctly on a NetBIOS service based on this RFC.

4.2. USE EXISTING STANDARDS

Protocol development, especially with standardization, is a demanding process. The development of new protocols must be minimized.

It is considered essential that an existing standard which provides the necessary functionality with reasonable performance always be chosen in preference to developing a new protocol.

When a standard protocol is used, it must be unmodified.

4.3. MINIMIZE OPTIONS

The standard for NetBIOS on TCP should contain few, if any, options.

Where options are included, the options should be designed so that devices with different option selections should interoperate.

4.4. TOLERATE ERRORS AND DISRUPTIONS

NetBIOS networks typically operate in an uncontrolled environment. Computers come on-line at arbitrary times. Computers usually go off-line without any notice to their peers. The software is often operated by users who are unfamiliar with networks and who may randomly perturb configuration settings.

Despite this chaos, NetBIOS networks work. NetBIOS on TCP must also

be able to operate well in this environment.

Robust operation does not necessarily mean that the network is proof against all disruptions. A typical NetBIOS network may be disrupted by certain types of behavior, whether inadvertent or malicious.

4.5. DO NOT REQUIRE CENTRAL MANAGEMENT

NetBIOS on TCP should be able to operate, if desired, without centralized management beyond that typically required by a TCP based network.

4.6. ALLOW INTERNET OPERATION

The proposed standard recognizes the need for NetBIOS operation across a set of networks interconnected by network (IP) level relays (gateways.)

However, the standard assumes that this form of operation will be less frequent than on the local MAC bridged-LAN.

4.7. MINIMIZE BROADCAST ACTIVITY

The standard pre-supposes that the only broadcast services are those supported by UDP. Multicast capabilities are not assumed to be available in any form.

Despite the availability of broadcast capabilities, the standard recognizes that some administrations may wish to avoid heavy broadcast activity. For example, an administration may wish to avoid isolated non-participating hosts from the burden of receiving and discarding NetBIOS broadcasts.

4.8. PERMIT IMPLEMENTATION ON EXISTING SYSTEMS

The NetBIOS on TCP protocol should be implementable on common operating systems, such as Unix(tm) and VAX/VMS(tm), without massive effort.

The NetBIOS protocols should not require services typically unavailable on presently existing TCP/UDP/IP implementations.

4.9. REQUIRE ONLY THE MINIMUM NECESSARY TO OPERATE

The protocol definition should specify only the minimal set of protocols required for interoperation. However, additional protocol elements may be defined to enhance efficiency. These latter elements may be generated at the option of the sender, although they must be accepted by all receivers.

4.10. MAXIMIZE EFFICIENCY

To be useful, a protocol must conduct its business quickly.

4.11. MINIMIZE NEW INVENTIONS

When an existing protocol is not quite able to support a necessary function, but with a small amount of change, it could, that protocol should be used. This is felt to be easier to achieve than development of new protocols; further, it is likely to have more general utility for the Internet.

5. OVERVIEW OF NetBIOS

This section describes the NetBIOS services. It is for background information only. The reader may chose to skip to the next section.

NetBIOS was designed for use by groups of PCs, sharing a broadcast medium. Both connection (Session) and connectionless (Datagram) services are provided, and broadcast and multicast are supported. Participants are identified by name. Assignment of names is distributed and highly dynamic.

NetBIOS applications employ NetBIOS mechanisms to locate resources, establish connections, send and receive data with an application peer, and terminate connections. For purposes of discussion, these mechanisms will collectively be called the NetBIOS Service.

This service can be implemented in many different ways. One of the first implementations was for personal computers running the PC-DOS and MS-DOS operating systems. It is possible to implement NetBIOS within other operating systems, or as processes which are, themselves, simply application programs as far as the host operating system is concerned.

The NetBIOS specification, published by IBM as "Technical Reference PC Network"[2] defines the interface and services available to the NetBIOS user. The protocols outlined by that document pertain only to the IBM PC Network and are not generally applicable to other networks.

5.1. INTERFACE TO APPLICATION PROGRAMS

NetBIOS on personal computers includes both a set of services and an exact program interface to those services. NetBIOS on other computer systems may present the NetBIOS services to programs using other interfaces. Except on personal computers, no clear standard for a NetBIOS software interface has emerged.

5.2. NAME SERVICE

NetBIOS resources are referenced by name. Lower-level address information is not available to NetBIOS applications. An application, representing a resource, registers one or more names that it wishes to use.

The name space is flat and uses sixteen alphanumeric characters. Names may not start with an asterisk (*).

Registration is a bid for use of a name. The bid may be for exclusive (unique) or shared (group) ownership. Each application contends with the other applications in real time. Implicit permission is granted to a station when it receives no objections. That is, a bid is made and the application waits for a period of time. If no objections are received, the station assumes that it has permission.

A unique name should be held by only one station at a time. However, duplicates ("name conflicts") may arise due to errors.

All instances of a group name are equivalent.

An application referencing a name generally does not know (or care) whether the name is registered as a unique or a group name.

An explicit name deletion function is specified, so that applications may remove a name. Implicit name deletion occurs when a station ceases operation. In the case of personal computers, implicit name deletion is a frequent occurrence.

The Name Service primitives are:

1) Add Name

   The requesting application wants exclusive use of the name.

2) Add Group Name

   The requesting application is willing to share use of the name with other applications.

3) Delete Name

   The application no longer requires use of the name. It is important to note that typical use of NetBIOS is among independently-operated personal computers. A common way to stop using a PC is to turn it off; in this case, the graceful give-back mechanism, provided by the Delete Name function, is not used. Because this occurs frequently, the network service must support this behavior.

5.3. SESSION SERVICE

A session is a reliable message exchange, conducted between a pair of NetBIOS applications. Sessions are full-duplex, sequenced, and reliable. Data is organized into messages. Each message may range in size from 0 to 131,071 bytes. No expedited or urgent data capabilities are present.

Multiple sessions may exist between any pair of calling and called names.

The parties to a connection have access to the calling and called names.

The NetBIOS specification does not define how a connection request to a shared (group) name resolves into a session. The usual assumption is that a session may be established with any one owner of the called group name.

An important service provided to NetBIOS applications is the detection of sessions failure. The loss of a session is reported to an application via all of the outstanding service requests for that session. For example, if the application has only a NetBIOS receive primitive pending and the session terminates, the pending receive will abort with a termination indication.

Session Service primitives are:

1) Call

   Initiate a session with a process that is listening under the specified name. The calling entity must indicate both a calling name (properly registered to the caller) and a called name.

2) Listen

   Accept a session from a caller. The listen primitive may be constrained to accept an incoming call from a named caller. Alternatively, a call may be accepted from any caller.

3) Hang Up

   Gracefully terminate a session. All pending data is transferred before the session is terminated.

4) Send

   Transmit one message. A time-out can occur. A time-out of any session send forces the non-graceful termination of the session.

A "chain send" primitive is required by the PC NetBIOS software interface to allow a single message to be gathered from pieces in various buffers. Chain Send is an interface detail and does not effect the protocol.

5) Receive

Receive data. A time-out can occur. A time-out on a session receive only terminates the receive, not the session, although the data is lost.

The receive primitive may be implemented with variants, such as "Receive Any", which is required by the PC NetBIOS software interface. Receive Any is an interface detail and does not effect the protocol.

6) Session Status

Obtain information about all of the requestor's sessions, under the specified name. No network activity is involved.

5.4. DATAGRAM SERVICE

The Datagram service is an unreliable, non-sequenced, connectionless service. Datagrams are sent under cover of a name properly registered to the sender.

Datagrams may be sent to a specific name or may be explicitly broadcast.

Datagrams sent to an exclusive name are received, if at all, by the holder of that name. Datagrams sent to a group name are multicast to all holders of that name. The sending application program cannot distinguish between group and unique names and thus must act as if all non-broadcast datagrams are multicast.

As with the Session Service, the receiver of the datagram is told the sending and receiving names.

Datagram Service primitives are:

1) Send Datagram

Send an unreliable datagram to an application that is associated with the specified name. The name may be unique or group; the sender is not aware of the difference. If the name belongs to a group, then each member is to receive the datagram.

2) Send Broadcast Datagram

Send an unreliable datagram to any application with a Receive Broadcast Datagram posted.

3) Receive Datagram

Receive a datagram sent by a specified originating name to the specified name. If the originating name is an asterisk, then the datagram may have been originated under any name.

Note: An arriving datagram will be delivered to all pending Receiving Datagrams that have source and destination specifications matching those of the datagram. In other words, if a program (or group of programs) issue a series of identical Receive Datagrams, one datagram will cause the entire series to complete.

4) Receive Broadcast Datagram

Receive a datagram sent as a broadcast.

If there are multiple pending Receive Broadcast Datagram operations pending, all will be satisfied by the same received datagram.

5.5. MISCELLANEOUS FUNCTIONS

The following functions are present to control the operation of the hardware interface to the network. These functions are generally implementation dependent.

1) Reset

Initialize the local network adapter.

2) Cancel

Abort a pending NetBIOS request. The successful cancel of a Send (or Chain Send) operation will terminate the associated session.

3) Adapter Status

Obtain information about the local network adapter or of a remote adapter.

4) Unlink

For use with Remote Program Load (RPL). Unlink redirects the PC boot disk device back to the local disk. See the

NetBIOS specification for further details concerning RPL and the Unlink operation (see page 2-35 in [2]).

5) Remote Program Load

Remote Program Load (RPL) is not a NetBIOS function. It is a NetBIOS application defined by IBM in their NetBIOS specification (see pages 2-80 through 2-82 in [2]).

5.6. NON-STANDARD EXTENSIONS

The IBM Token Ring implementation of NetBIOS has added at least one new user capability:

1) Find Name

This function determines whether a given name has been registered on the network.

6. NetBIOS FACILITIES SUPPORTED BY THIS STANDARD

The protocol specified by this standard permits an implementer to provide all of the NetBIOS services as described in the IBM "Technical Reference PC Network"[2].

The following NetBIOS facilities are outside the scope of this specification. These are local implementation matters and do not impact interoperability:

- RESET
- SESSION STATUS
- UNLINK
- RPL (Remote Program Load)

7. REQUIRED SUPPORTING SERVICE INTERFACES AND DEFINITIONS

The protocols described in this RFC require service interfaces to the following:

- TCP[3,4]
- UDP[5]

Byte ordering, addressing conventions (including addresses to be used for broadcasts and multicasts) are defined by the most recent version of:

- Assigned Numbers[6]

Additional definitions and constraints are in:

- IP[7]
- Internet Subnets[8,9,10]

8. RELATED PROTOCOLS AND SERVICES

The design of the protocols described in this RFC allow for the future incorporation of the following protocols and services. However, before this may occur, certain extensions may be required to the protocols defined in this RFC or to those listed below.

- Domain Name Service[11,12,13,14]
- Internet Group Multicast[15,16]

9. NetBIOS SCOPE

A "NetBIOS Scope" is the population of computers across which a registered NetBIOS name is known. NetBIOS broadcast and multicast datagram operations must reach the entire extent of the NetBIOS scope.

An internet may support multiple, non-intersecting NetBIOS Scopes.

Each NetBIOS scope has a "scope identifier". This identifier is a character string meeting the requirements of the domain name system for domain names.

NOTE: Each implementation of NetBIOS-over-TCP must provide mechanisms to manage the scope identifier(s) to be used.

Control of scope identifiers implies a requirement for additional NetBIOS interface capabilities. These may be provided through extensions of the user service interface or other means (such as node configuration parameters.) The nature of these extensions is not part of this specification.

10. NetBIOS END-NODES

End-nodes support NetBIOS service interfaces and contain applications.

Three types of end-nodes are part of this standard:

- Broadcast ("B") nodes
- Point-to-point ("P") nodes
- Mixed mode ("M") nodes

An IP address may be associated with only one instance of one of the above types.

Without having preloaded name-to-address tables, NetBIOS participants

are faced with the task of dynamically resolving references to one another. This can be accomplished with broadcast or mediated point-to-point communications.

B nodes use local network broadcasting to effect a rendezvous with one or more recipients. P and M nodes use the NetBIOS Name Server (NBNS) and the NetBIOS Datagram Distribution Server (NBDD) for this same purpose.

End-nodes may be combined in various topologies. No matter how combined, the operation of the B, P, and M nodes is not altered.

NOTE: It is recommended that the administration of a NetBIOS scope avoid using both M and B nodes within the same scope. A NetBIOS scope should contain only B nodes or only P and M nodes.

10.1. BROADCAST (B) NODES

Broadcast (or "B") nodes communicate using a mix of UDP datagrams (both broadcast and directed) and TCP connections. B nodes may freely interoperate with one another within a broadcast area. A broadcast area is a single MAC-bridged "B-LAN". (See Appendix A for a discussion of using Internet Group Multicasting as a means to extend a broadcast area beyond a single B-LAN.)

10.2. POINT-TO-POINT (P) NODES

Point-to-point (or "P") nodes communicate using only directed UDP datagrams and TCP sessions. P nodes neither generate nor listen for broadcast UDP packets. P nodes do, however, offer NetBIOS level broadcast and multicast services using capabilities provided by the NBNS and NBDD.

P nodes rely on NetBIOS name and datagram distribution servers. These servers may be local or remote; P nodes operate the same in either case.

10.3. MIXED MODE (M) NODES

Mixed mode nodes (or "M") nodes are P nodes which have been given certain B node characteristics. M nodes use both broadcast and unicast. Broadcast is used to improve response time using the assumption that most resources reside on the local broadcast medium rather than somewhere in an internet.

M nodes rely upon NBNS and NBDD servers. However, M nodes may continue limited operation should these servers be temporarily unavailable.

11. NetBIOS SUPPORT SERVERS

Two types of support servers are part of this standard:

- NetBIOS name server ("NBNS") nodes
- Netbios datagram distribution ("NBDD") nodes

NBNS and NBDD nodes are invisible to NetBIOS applications and are part of the underlying NetBIOS mechanism.

NetBIOS name and datagram distribution servers are the focus of name and datagram activity for P and M nodes.

Both the name (NBNS) and datagram distribution (NBDD) servers are permitted to shift part of their operation to the P or M end-node which is requesting a service.

Since the assignment of responsibility is dynamic, and since P and M nodes must be prepared to operate should the NetBIOS server delegate control to the maximum extent, the system naturally accommodates improvements in NetBIOS server function. For example, as Internet Group Multicasting becomes more widespread, new NBDD implementations may elect to assume full responsibility for NetBIOS datagram distribution.

Interoperability between different implementations is assured by imposing requirements on end-node implementations that they be able to accept the full range of legal responses from the NBNS or NBDD.

11.1. NetBIOS NAME SERVER (NBNS) NODES

The NBNS is designed to allow considerable flexibility with its degree of responsibility for the accuracy and management of NetBIOS names. On one hand, the NBNS may elect not to accept full responsibility, leaving the NBNS essentially a "bulletin board" on which name/address information is freely posted (and removed) by P and M nodes without validation by the NBNS. Alternatively, the NBNS may elect to completely manage and validate names. The degree of responsibility that the NBNS assumes is asserted by the NBNS each time a name is claimed through a simple mechanism. Should the NBNS not assert full control, the NBNS returns enough information to the requesting node so that the node may challenge any putative holder of the name.

This ability to shift responsibility for NetBIOS name management between the NBNS and the P and M nodes allows a network administrator (or vendor) to make a tradeoff between NBNS simplicity, security, and delay characteristics.

A single NBNS may be implemented as a distributed entity, such as the Domain Name Service. However, this RFC does not attempt to define

the internal communications which would be used.

11.1.1. RELATIONSHIP OF THE NBNS TO THE DOMAIN NAME SYSTEM

The NBNS design attempts to align itself with the Domain Name System in a number of ways.

First, the NetBIOS names are encoded in a form acceptable to the domain name system.

Second, a scope identifier is appended to each NetBIOS name. This identifier meets the restricted character set of the domain system and has a leading period. This makes the NetBIOS name, in conjunction with its scope identifier, a valid domain system name.

Third, the negotiated responsibility mechanisms permit the NBNS to be used as a simple bulletin board on which are posted (name,address) pairs. This parallels the existing domain sytem query service.

This RFC, however, requires the NBNS to provide services beyond those provided by the current domain name system. An attempt has been made to coalesce all the additional services which are required into a set of transactions which follow domain name system styles of interaction and packet formats.

Among the areas in which the domain name service must be extended before it may be used as an NBNS are:

- Dynamic addition of entries
- Dynamic update of entry data
- Support for multiple instance (group) entries
- Support for entry time-to-live values and ability to accept refresh messages to restart the time-to-live period
- New entry attributes

11.2. NetBIOS DATAGRAM DISTRIBUTION SERVER (NBDD) NODES

The internet does not yet support broadcasting or multicasting. The NBDD extends NetBIOS datagram distribution service to this environment.

The NBDD may elect to complete, partially complete, or totally refuse to service a node’s request to distribute a NetBIOS datagram. An end-node may query an NBDD to determine whether the NBDD will deliver a datagram to a specific NetBIOS name.

The design of NetBIOS-over-TCP lends itself to the use of Internet Group Multicast. For details see Appendix A.

11.3. RELATIONSHIP OF NBNS AND NBDD NODES

This RFC defines the NBNS and NBDD as distinct, separate entities.

In the absence of NetBIOS name information, a NetBIOS datagram distribution server must send a copy to each end-node within a NetBIOS scope.

An implementer may elect to construct NBNS and NBDD nodes which have a private protocol for the exchange of NetBIOS name information. Alternatively, an NBNS and NBDD may be implemented within the same device.

NOTE: Implementations containing private NBNS-NBDD protocols or combined NBNS-NBDD functions must be clearly identified.

11.4. RELATIONSHIP OF NetBIOS SUPPORT SERVERS AND B NODES

As defined in this RFC, neither NBNS nor NBDD nodes interact with B nodes. NetBIOS servers do not listen to broadcast traffic on any broadcast area to which they may be attached. Nor are the NetBIOS support servers even aware of B node activities or names claimed or used by B nodes.

It may be possible to extend both the NBNS and NBDD so that they participate in B node activities and act as a bridge to P and M nodes. However, such extensions are beyond the scope of this specification.

12. TOPOLOGIES

B, P, M, NBNS, and NBDD nodes may be combined in various ways to form useful NetBIOS environments. This section describes some of these combinations.

There are three classes of operation:

- Class 0: B nodes only.
- Class 1: P nodes only.
- Class 2: P and M nodes together.

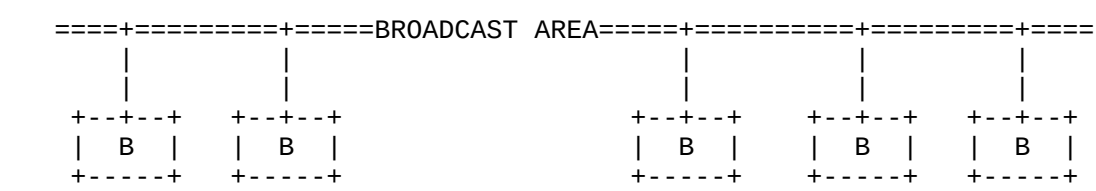
In the drawings which follow, any P node may be replaced by an M node. The effects of such replacement will be mentioned in conjunction with each example below.

12.1. LOCAL

A NetBIOS scope is operating locally when all entities are within the same broadcast area.

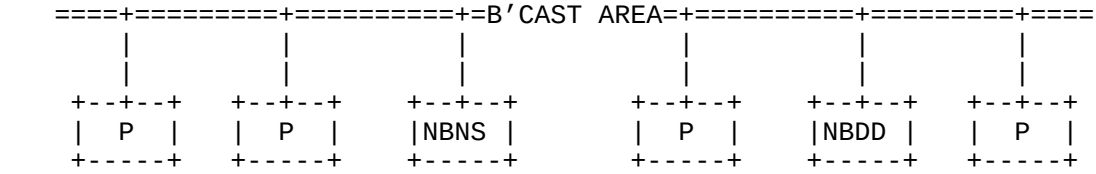
12.1.1. B NODES ONLY

Local operation with only B nodes is the most basic mode of operation. Name registration and discovery procedures use broadcast mechanisms. The NetBIOS scope is limited by the extent of the broadcast area. This configuration does not require NetBIOS support servers.



12.1.2. P NODES ONLY

This configuration would typically be used when the network administrator desires to eliminate NetBIOS as a source of broadcast activity.



This configuration operates the same as if it were in an internet and is cited here only due to its convenience as a means to reduce the use of broadcast.

Replacement of one or more of the P nodes with M nodes will not affect the operation of the other P and M nodes. P and M nodes will be able to interact with one another. Because M nodes use broadcast, overall broadcast activity will increase.

12.1.3. MIXED B AND P NODES

B and P nodes do not interact with one another. Replacement of P nodes with M nodes will allow B's and M's to interact.

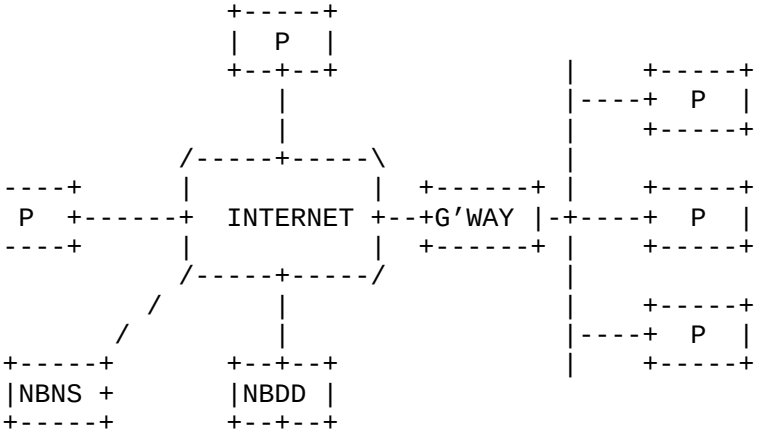
NOTE: B nodes and M nodes may be intermixed only on a local broadcast area. B and M nodes should not be intermixed in an internet environment.

12.2. INTERNET

12.2.1. P NODES ONLY

P nodes may be scattered at various locations in an internetwork. They require both an NBNS and an NBDD for NetBIOS name and datagram support, respectively.

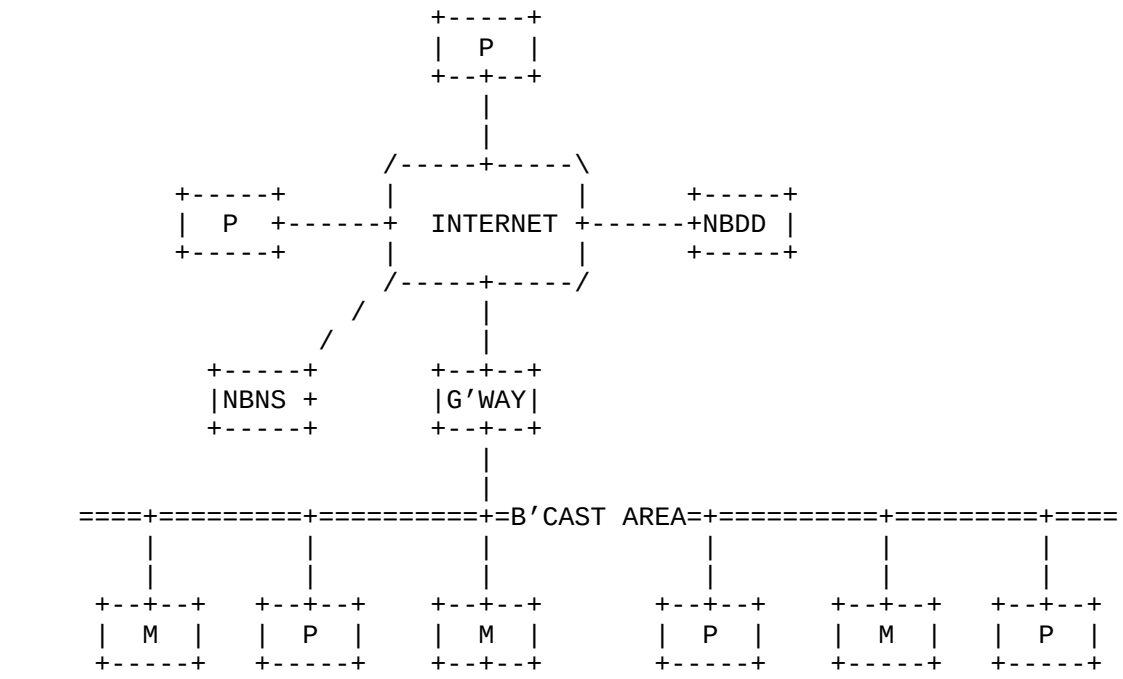
The NetBIOS scope is determined by the NetBIOS scope identifier (domain name) used by the various P (and M) nodes. An internet may contain numerous NetBIOS scopes.



Any P node may be replaced by an M node with no loss of function to any node. However, broadcast activity will be increased in the broadcast area to which the M node is attached.

12.2.2. MIXED M AND P NODES

M and P nodes may be mixed. When locating NetBIOS names, M nodes will tend to find names held by other M nodes on the same common broadcast area in preference to names held by P nodes or M nodes elsewhere in the network.



NOTE: B and M nodes should not be intermixed in an internet environment. Doing so would allow undetected NetBIOS name conflicts to arise and cause unpredictable behavior.

13. GENERAL METHODS

Overlying the specific protocols, described later, are a few general methods of interaction between entities.

13.1. REQUEST/RESPONSE INTERACTION STYLE

Most interactions between entities consist of a request flowing in one direction and a subsequent response flowing in the opposite direction.

In those situations where interactions occur on unreliable transports (i.e. UDP) or when a request is broadcast, there may not be a strict interlocking or one-to-one relationship between requests and responses.

In no case, however, is more than one response generated for a received request. While a response is pending the responding entity may send one or more wait acknowledgements.

13.1.1. RETRANSMISSION OF REQUESTS

UDP is an unreliable delivery mechanism where packets can be lost, received out of transmit sequence, duplicated and delivery can be significantly delayed. Since the NetBIOS protocols make heavy use of UDP, they have compensated for its unreliability with extra mechanisms.

Each NetBIOS packet contains all the necessary information to process it. None of the protocols use multiple UDP packets to convey a single request or response. If more information is required than will fit in a single UDP packet, for example, when a P-type node wants all the owners of a group name from a NetBIOS server, a TCP connection is used. Consequently, the NetBIOS protocols will not fail because of out of sequence delivery of UDP packets.

To overcome the loss of a request or response packet, each request operation will retransmit the request if a response is not received within a specified time limit.

Protocol operations sensitive to successive response packets, such as name conflict detection, are protected from duplicated packets because they ignore successive packets with the same NetBIOS information. Since no state on the responder’s node is associated with a request, the responder just sends the appropriate response whenever a request packet arrives. Consequently, duplicate or delayed request packets have no impact.

For all requests, if a response packet is delayed too long another request packet will be transmitted. A second response packet being sent in response to the second request packet is equivalent to a duplicate packet. Therefore, the protocols will ignore the second packet received. If the delivery of a response is delayed until after the request operation has been completed, successfully or not, the response packet is ignored.

13.1.2. REQUESTS WITHOUT RESPONSES: DEMANDS

Some request types do not have matching responses. These requests are known as "demands". In general a "demand" is an imperative request; the receiving node is expected to obey. However, because demands are unconfirmed, they are used only in situations where, at most, limited damage would occur if the demand packet should be lost.

Demand packets are not retransmitted.

13.2. TRANSACTIONS

Interactions between a pair of entities are grouped into "transactions". These transactions comprise one or more request/response pairs.

13.2.1. TRANSACTION ID

Since multiple simultaneous transactions may be in progress between a pair of entities a "transaction id" is used.

The originator of a transaction selects an ID unique to the originator. The transaction id is reflected back and forth in each interaction within the transaction. The transaction partners must match responses and requests by comparison of the transaction ID and the IP address of the transaction partner. If no matching request can be found the response must be discarded.

A new transaction ID should be used for each transaction. A simple 16 bit transaction counter ought to be an adequate id generator. It is probably not necessary to search the space of outstanding transaction ID to filter duplicates: it is extremely unlikely that any transaction will have a lifetime that is more than a small fraction of the typical counter cycle period. Use of the IP addresses in conjunction with the transaction ID further reduces the possibility of damage should transaction IDs be prematurely re-used.

13.3. TCP AND UDP FOUNDATIONS

This version of the NetBIOS-over-TCP protocols uses UDP for many interactions. In the future this RFC may be extended to permit such interactions to occur over TCP connections (perhaps to increase efficiency when multiple interactions occur within a short time or when NetBIOS datagram traffic reveals that an application is using NetBIOS datagrams to support connection- oriented service.)

14. REPRESENTATION OF NETBIOS NAMES

NetBIOS names as seen across the client interface to NetBIOS are exactly 16 bytes long. Within the NetBIOS-over-TCP protocols, a longer representation is used.

There are two levels of encoding. The first level maps a NetBIOS name into a domain system name. The second level maps the domain system name into the "compressed" representation required for interaction with the domain name system.

Except in one packet, the second level representation is the only NetBIOS name representation used in NetBIOS-over-TCP packet formats. The exception is the RDATA field of a NODE STATUS RESPONSE packet.

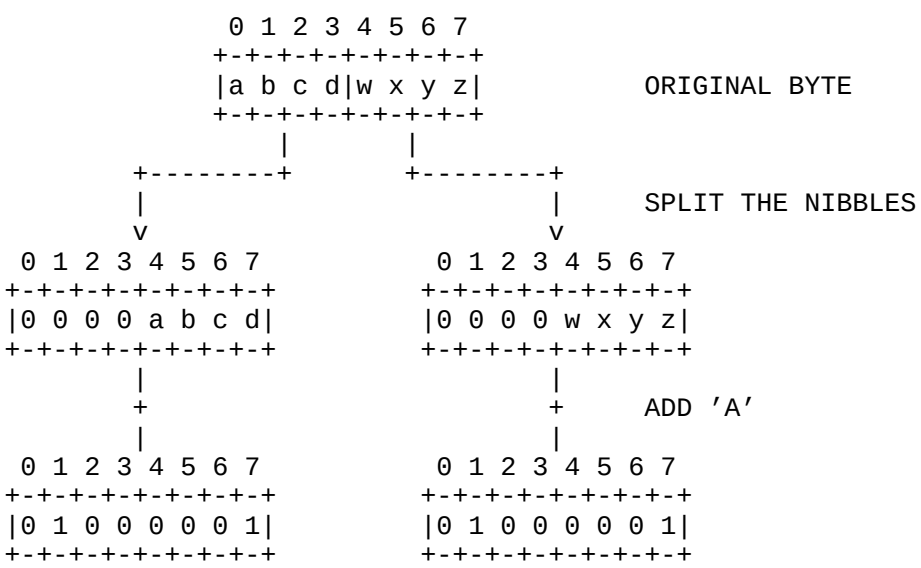
14.1. FIRST LEVEL ENCODING

The first level representation consists of two parts:

- NetBIOS name
- NetBIOS scope identifier

The 16 byte NetBIOS name is mapped into a 32 byte wide field using a reversible, half-ASCII, biased encoding. Each half-octet of the NetBIOS name is encoded into one byte of the 32 byte field. The first half octet is encoded into the first byte, the second half-octet into the second byte, etc.

Each 4-bit, half-octet of the NetBIOS name is treated as an 8-bit, right-adjusted, zero-filled binary number. This number is added to value of the ASCII character 'A' (hexidecimal 41). The resulting 8-bit number is stored in the appropriate byte. The following diagram demonstrates this procedure:



This encoding results in a NetBIOS name being represented as a sequence of 32 ASCII, upper-case characters from the set {A,B,C...N,O,P}.

The NetBIOS scope identifier is a valid domain name (without a leading dot).

An ASCII dot (2E hexidecimal) and the scope identifier are appended to the encoded form of the NetBIOS name, the result forming a valid domain name.

For example, the NetBIOS name "The NetBIOS name" in the NetBIOS scope "SCOPE.ID.COM" would be represented at level one by the ASCII character string:

FEGHGFCAEOGFHEECEJEPFDCAHEGBGNGF.SCOPE.ID.COM

14.2. SECOND LEVEL ENCODING

The first level encoding must be reduced to second level encoding. This is performed according to the rules defined in on page 31 of RFC 883[12] in the section on "Domain name representation and compression". Also see the section titled "Name Formats" in the Detailed Specifications[1].

15. NetBIOS NAME SERVICE

Before a name may be used, the name must be registered by a node. Once acquired, the name must be defended against inconsistent registration by other nodes. Before building a NetBIOS session or sending a NetBIOS datagram, the one or more holders of the name must be located.

The NetBIOS name service is the collection of procedures through which nodes acquire, defend, and locate the holders of NetBIOS names.

The name service procedures are different depending whether the end-node is of type B, P, or M.

15.1. OVERVIEW OF NetBIOS NAME SERVICE

15.1.1. NAME REGISTRATION (CLAIM)

Each NetBIOS node can own more than one name. Names are acquired dynamically through the registration (name claim) procedures.

Every node has a permanent unique name. This name, like any other name, must be explicitly registered by all end-node types.

A name can be unique (exclusive) or group (non-exclusive). A unique name may be owned by a single node; a group name may be owned by any number of nodes. A name ceases to exist when it is not owned by at least one node. There is no intrinsic quality of a name which determines its characteristics: these are established at the time of registration.

Each node maintains state information for each name it has registered. This information includes:

- Whether the name is a group or unique name
- Whether the name is "in conflict"
- Whether the name is in the process of being deleted

B nodes perform name registration by broadcasting claim requests, soliciting a defense from any node already holding the name.

P nodes perform name registration through the agency of the NBNS.

M nodes register names through an initial broadcast, like B nodes, then, in the absence of an objection, by following the same procedures as a P node. In other words, the broadcast action may terminate the attempt, but is not sufficient to confirm the registration.

15.1.2. NAME QUERY (DISCOVERY)

Name query (also known as "resolution" or "discovery") is the procedure by which the IP address(es) associated with a NetBIOS name are discovered. Name query is required during the following operations:

During session establishment, calling and called names must be specified. The calling name must exist on the node that posts the CALL. The called name must exist on a node that has previously posted a LISTEN. Either name may be a unique or group name.

When a directed datagram is sent, a source and destination name must be specified. If the destination name is a group name, a datagram is sent to all the members of that group.

Different end-node types perform name resolution using different techniques, but using the same packet formats:

- B nodes solicit name information by broadcasting a request.

- P nodes ask the NBNS.

- M nodes broadcast a request. If that does not provide the desired information, an inquiry is sent to the NBNS.

15.1.3. NAME RELEASE

NetBIOS names may be released explicitly or silently by an end- node. Silent release typically occurs when an end-node fails or is turned-off. Most of the mechanisms described below are present to detect silent name release.

15.1.3.1. EXPLICIT RELEASE

B nodes explicitly release a name by broadcasting a notice.

P nodes send a notification to their NBNS.

M nodes both broadcast a notice and inform their supporting NBNS.

15.1.3.2. NAME LIFETIME AND REFRESH

Names held by an NBNS are given a lifetime during name registration. The NBNS will consider a name to have been silently released if the end-node fails to send a name refresh message to the NBNS before the lifetime expires. A refresh restarts the lifetime clock.

NOTE: The implementor should be aware of the tradeoff between accuracy of the database and the internet overhead that the refresh mechanism introduces. The lifetime period should be tuned accordingly.

For group names, each end-node must send refresh messages. A node that fails to do so will be considered to have silently released the name and dropped from the group.

The lifetime period is established through a simple negotiation mechanism during name registration: In the name registration request, the end-node proposes a lifetime value or requests an infinite lifetime. The NBNS places an actual lifetime value into the name registration response. The NBNS is always allowed to respond with an infinite actual period. If the end node proposed an infinite lifetime, the NBNS may respond with any definite period. If the end node proposed a definite period, the NBNS may respond with any definite period greater than or equal to that proposed.

This negotiation of refresh times gives the NBNS means to disable or enable refresh activity. The end-nodes may set a minimum refresh cycle period.

NBNS implementations which are completely reliable may disable refresh.

15.1.3.3. NAME CHALLENGE

To detect whether a node has silently released its claim to a name, it is necessary on occasion to challenge that node’s current ownership. If the node defends the name then the node is allowed to continue possession. Otherwise it is assumed that the node has released the name.

A name challenge may be issued by an NBNS or by a P or M node. A challenge may be directed towards any end-node type: B, P, or M.

15.1.3.4. GROUP NAME FADE-OUT

NetBIOS groups may contain an arbitrarily large number of members. The time to challenge all members could be quite large.

To avoid long delays when names are claimed through an NBNS, an

optimistic heuristic has been adopted. It is assumed that there will always be some node which will defend a group name. Consequently, it is recommended that the NBNS will immediately reject a claim request for a unique name when there already exists a group with the same name. The NBNS will never return an IP address (in response to a NAME REGISTRATION REQUEST) when a group name exists.

An NBNS will consider a group to have faded out of existence when the last remaining member fails to send a timely refresh message or explicitly releases the name.

15.1.3.5. NAME CONFLICT

Name conflict exists when a unique name has been claimed by more than one node on a NetBIOS network. B, M, and NBNS nodes may detect a name conflict. The detection mechanism used by B and M nodes is active only during name discovery. The NBNS may detect conflict at any time it verifies the consistency of its name database.

B and M nodes detect conflict by examining the responses received in answer to a broadcast name query request. The first response is taken as authoritative. Any subsequent, inconsistent responses represent conflicts.

Subsequent responses are inconsistent with the authoritative response when:

```
     The subsequent response has the same transaction ID as the
     NAME QUERY REQUEST.
  AND
     The subsequent response is not a duplicate of the
     authoritative response.
  AND EITHER:
          The group/unique characteristic of the authoritative
          response is "unique".
       OR
          The group/unique characteristic of the subsequent
          response is "unique".
```

The period in which B and M nodes examine responses is limited by a conflict timer, CONFLICT_TIMER. The accuracy or duration of this timer is not crucial: the NetBIOS system will continue to operate even with persistent name conflicts.

Conflict conditions are signaled by sending a NAME CONFLICT DEMAND to the node owning the offending name. Nothing is sent to the node which originated the authoritative response.

Any end-node that receives NAME CONFLICT DEMAND is required to update its "local name table" to reflect that the name is in conflict. (The "local name table" on each node contains names that have been

successfully registered by that node.)

Notice that only those nodes that receive the name conflict message place a conflict mark next to a name.

Logically, a marked name does not exist on that node. This means that the node should not defend the name (for name claim purposes), should not respond to a name discovery requests for that name, nor should the node send name refresh messages for that name. Furthermore, it can no longer be used by that node for any session establishment or sending or receiving datagrams. Existing sessions are not affected at the time a name is marked as being in conflict.

The only valid user function against a marked name is DELETE NAME. Any other user NetBIOS function returns immediately with an error code of "NAME CONFLICT".

15.1.4. ADAPTER STATUS

An end-node or the NBNS may ask any other end-node for a collection of information about the NetBIOS status of that node. This status consists of, among other things, a list of the names which the node believes it owns. The returned status is filtered to contain only those names which have the same NetBIOS scope identifier as the requestor’s name.

When requesting node status, the requestor identifies the target node by NetBIOS name A name query transaction may be necessary to acquire the IP address for the name. Locally cached name information may be used in lieu of a query transaction. The requesting node sends a NODE STATUS REQUEST. In response, the receiving node sends a NODE STATUS RESPONSE containing its local name table and various statistics.

The amount of status which may be returned is limited by the size of a UDP packet. However, this is sufficient for the typical NODE STATUS RESPONSE packet.

15.1.5. END-NODE NBNS INTERACTION

There are certain characteristics of end-node to NBNS interactions which are in common and are independent of any particular transaction type.

15.1.5.1. UDP, TCP, AND TRUNCATION

For all transactions between an end-node and an NBNS, either UDP or TCP may be used as a transport. If the NBNS receives a UDP based request, it will respond using UDP. If the amount of information exceeds what fits into a UDP packet, the response will contain a "truncation flag". In this situation, the end- node may open a TCP

connection to the NBNS, repeat the request, and receive a complete, untruncated response.

15.1.5.2. NBNS WACK

While a name service request is in progress, the NBNS may issue a WAIT FOR ACKNOWLEDGEMENT RESPONSE (WACK) to assure the client end-node that the NBNS is still operational and is working on the request.

15.1.5.3. NBNS REDIRECTION

The NBNS, because it follows Domain Name system styles of interaction, is permitted to redirect a client to another NBNS.

15.1.6. SECURED VERSUS NON-SECURED NBNS

An NBNS may be implemented in either of two general ways: The NBNS may monitor, and participate in, name activity to ensure consistency. This would be a "secured" style NBNS. Alternatively, an NBNS may be implemented to be essentially a "bulletin board" on which name information is posted and responsibility for consistency is delegated to the end-nodes. This would be a "non-secured" style NBNS.

15.1.7. CONSISTENCY OF THE NBNS DATA BASE

Even in a properly running NetBIOS scope the NBNS and its community of end-nodes may occasionally lose synchronization with respect to the true state of name registrations.

This may occur should the NBNS fail and lose all or part of its database.

More commonly, a P or M node may be turned-off (thus forgetting the names it has registered) and then be subsequently turned back on.

Finally, errors may occur or an implementation may be incorrect.

Various approaches have been incorporated into the NetBIOS-over- TCP protocols to minimize the impact of these problems.

1. The NBNS (or any other node) may "challenge" (using a NAME QUERY REQUEST) an end-node to verify that it actually owns a name.

   Such a challenge may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond causes the NBNS to consider that the end-node has released the name in question.

(If UDP is being used as the underlying transport, the challenge, like all other requests, must be retransmitted some number of times in the absence of a response.)

2. The NBNS (or any other node) may request (using the NODE STATUS REQUEST) that an end-node deliver its entire name table.

   This may occur at any time. Every end-node must be prepared to make a timely response.

   Failure to respond permits (but does not require) the NBNS to consider that the end-node has failed and released all names to which it had claims. (Like the challenge, on a UDP transport, the request must be retransmitted in the absence of a response.)

3. The NBNS may revoke a P or M node's use of a name by sending either a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the node.

   The receiving end-node may continue existing sessions which use that name, but must otherwise cease using that name. If the NBNS placed the name in conflict, the name may be re-acquired only by deletion and subsequent reclamation. If the NBNS requested that the name be released, the node may attempt to re-acquire the name without first performing a name release transaction.

4. The NBNS may impose a "time-to-live" on each name it registers. The registering node is made aware of this time value during the name registration procedure.

   Simple or reliable NBNS's may impose an infinite time-to-live.

5. If an end-node holds any names that have finite time-to-live values, then that node must periodically send a status report to the NBNS. Each name is reported using the NAME REFRESH REQUEST packet.

   These status reports restart the timers of both the NBNS and the reporting node. However, the only timers which are restarted are those associated with the name found in the status report. Timers on other names are not affected.

   The NBNS may consider that a node has released any name which has not been refreshed within some multiple of name's time-to-live.

   A well-behaved NBNS, would, however, issue a challenge to-,

or request a list of names from-, the non-reporting end-node before deleting its name(s). The absence of a response, or of the name in a response, will confirm the NBNS decision to delete a name.

6. The absence of reports may cause the NBNS to infer that the end-node has failed. Similarly, receipt of information widely divergent from what the NBNS believes about the node, may cause the NBNS to consider that the end-node has been restarted.

   The NBNS may analyze the situation through challenges or requests for a list of names.

7. A very cautious NBNS is free to poll nodes (by sending NAME QUERY REQUEST or NODE STATUS REQUEST packets) to verify that their name status is the same as that registered in the NBNS.

   NOTE: Such polling activity, if used at all by an implementation, should be kept at a very low level or enabled only during periods when the NBNS has some reason to suspect that its information base is inaccurate.

8. P and M nodes can detect incorrect name information at session establishment.

   If incorrect information is found, NBNS is informed via a NAME RELEASE REQUEST originated by the end-node which detects the error.

15.1.8. NAME CACHING

An end-node may keep a local cache of NetBIOS name-to-IP address translation entries.

All cache entries should be flushed on a periodic basis.

In addition, a node ought to flush any cache information associated with an IP address if the node receives any information indicating that there may be any possibility of trouble with the node at that IP address. For example, if a NAME CONFLICT DEMAND is sent to a node, all cached information about that node should be cleared within the sending node.

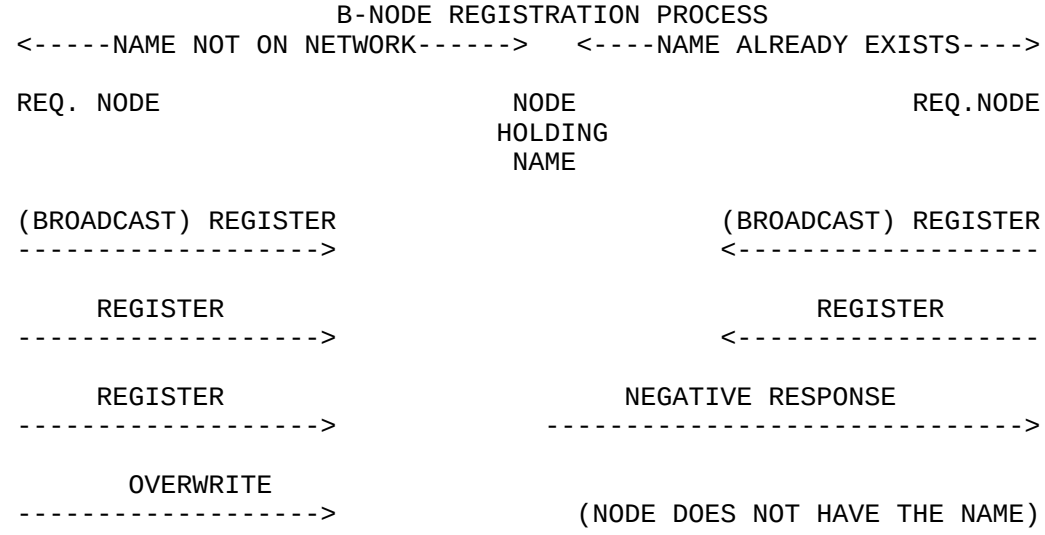
15.2. NAME REGISTRATION TRANSACTIONS

15.2.1. NAME REGISTRATION BY B NODES

A name claim transaction initiated by a B node is broadcast throughout the broadcast area. The NAME REGISTRATION REQUEST will be

heard by all B and M nodes in the area. Each node examines the claim to see whether it it is consistent with the names it owns. If an inconsistency exists, a NEGATIVE NAME REGISTRATION RESPONSE is unicast to the requestor. The requesting node obtains ownership of the name (or membership in the group) if, and only if, no NEGATIVE NAME REGISTRATION RESPONSEs are received within the name claim timeout, CONFLICT_TIMER. (See "Defined Constants and Variables" in the Detailed Specification for the value of this timer.)

A B node proclaims its new ownership by broadcasting a NAME OVERWRITE DEMAND.



The NAME REGISTRATION REQUEST, like any request, must be repeated if no response is received within BCAST_REQ_RETRY_TIMEOUT. Transmission of the request is attempted BCAST_REQ_RETRY_COUNT times.

15.2.2. NAME REGISTRATION BY P NODES

A name registration may proceed in various ways depending whether the name being registered is new to the NBNS. If the name is known to the NBNS, then challenges may be sent to the prior holder(s).

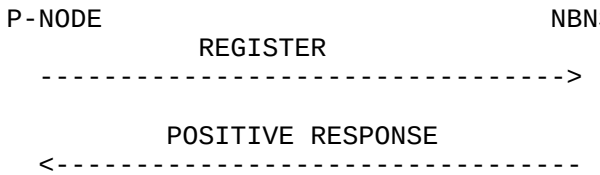
15.2.2.1. NEW NAME, OR NEW GROUP MEMBER

The diagram, below, shows the sequence of events when an end-node registers a name which is new to the NBNS. (The diagram omits WACKs, NBNS redirections, and retransmission of requests.)

This same interaction will occur if the name being registered is a group name and the group already exists. The NBNS will add the

registrant to the set of group members.

P-NODE REGISTRATION PROCESS
(server has no previous information about the name)



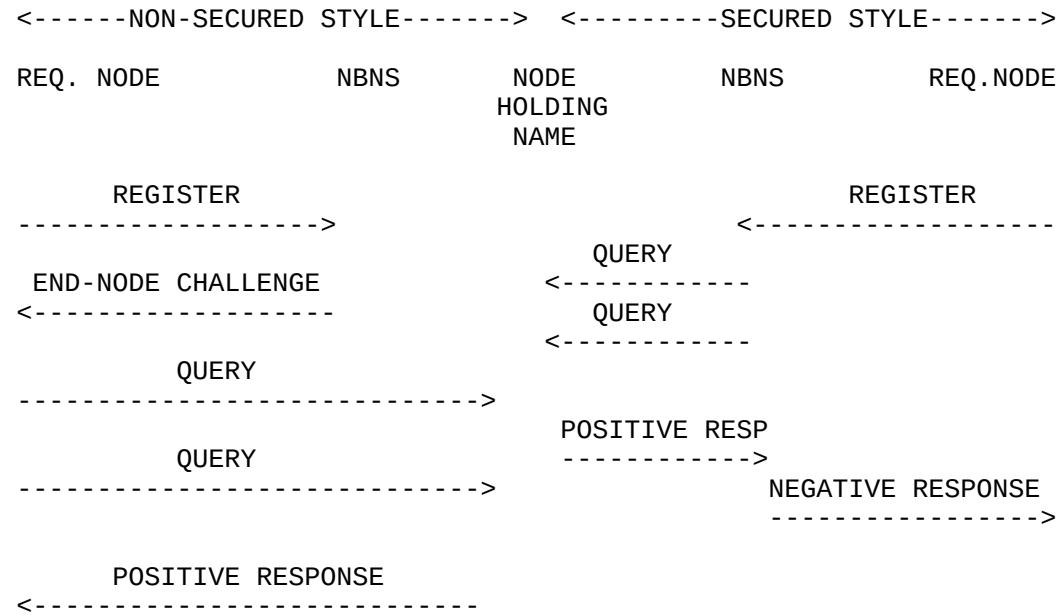
The interaction is rather simple: the end-node sends a NAME REGISTRATION REQUEST, the NBNS responds with a POSITIVE NAME REGISTRATION RESPONSE.

15.2.2.2. EXISTING NAME AND OWNER IS STILL ACTIVE

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is still active.

There are two sides to the diagram. The left side shows how a non-secured NBNS would handle the matter. Secured NBNS activity is shown on the right.

P-NODE REGISTRATION PROCESS
(server HAS a previous owner that IS active)



A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will defend against the challenge and the registering end-node will simply drop the registration attempt without further interaction with the NBNS.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is still being defended and consequently returns a NEGATIVE NAME REGISTRATION RESPONSE to the registrant.

Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

Although not shown in the diagram, a non-secured NBNS will send a NEGATIVE NAME REGISTRATION RESPONSE to a request to register a unique name when there already exists a group of the same name. A secured NBNS may elect to poll (or challenge) the group members to determine whether any active members remain. This may impose a heavy load on the network. It is recommended that group names be allowed to fade-out through the name refresh mechanism.

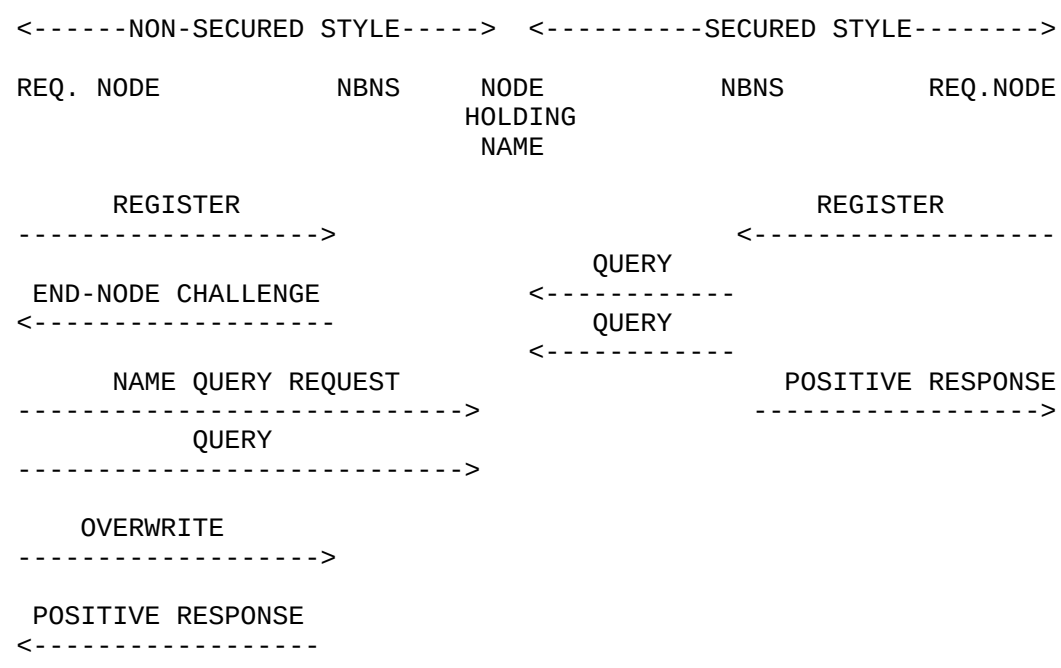
15.2.2.3. EXISTING NAME AND OWNER IS INACTIVE

The following diagram shows interactions when an attempt is made to register a unique name, the NBNS is aware of an existing owner, and that existing owner is no longer active.

A non-secured NBNS will answer the NAME REGISTRATION REQUEST with a END-NODE CHALLENGE REGISTRATION RESPONSE. This response asks the end-node to issue a challenge transaction against the node indicated in the response. In this case, the prior node will not defend against the challenge. The registrant will inform the NBNS through a NAME OVERWRITE REQUEST. The NBNS will replace the prior name information in its database with the information in the overwrite request.

A secured NBNS will refrain from answering the NAME REGISTRATION REQUEST until the NBNS has itself challenged the prior holder(s) of the name. In this case, the NBNS finds that that the name is not being defended and consequently returns a POSITIVE NAME REGISTRATION RESPONSE to the registrant.

P-NODE REGISTRATION PROCESS
(server HAS a previous owner that is NOT active)



Due to the potential time for the secured NBNS to make the challenge(s), it is likely that a WACK will be sent by the NBNS to the registrant.

A secured NBNS will immediately send a NEGATIVE NAME REGISTRATION RESPONSE in answer to any NAME OVERWRITE REQUESTS it may receive.

15.2.3. NAME REGISTRATION BY M NODES

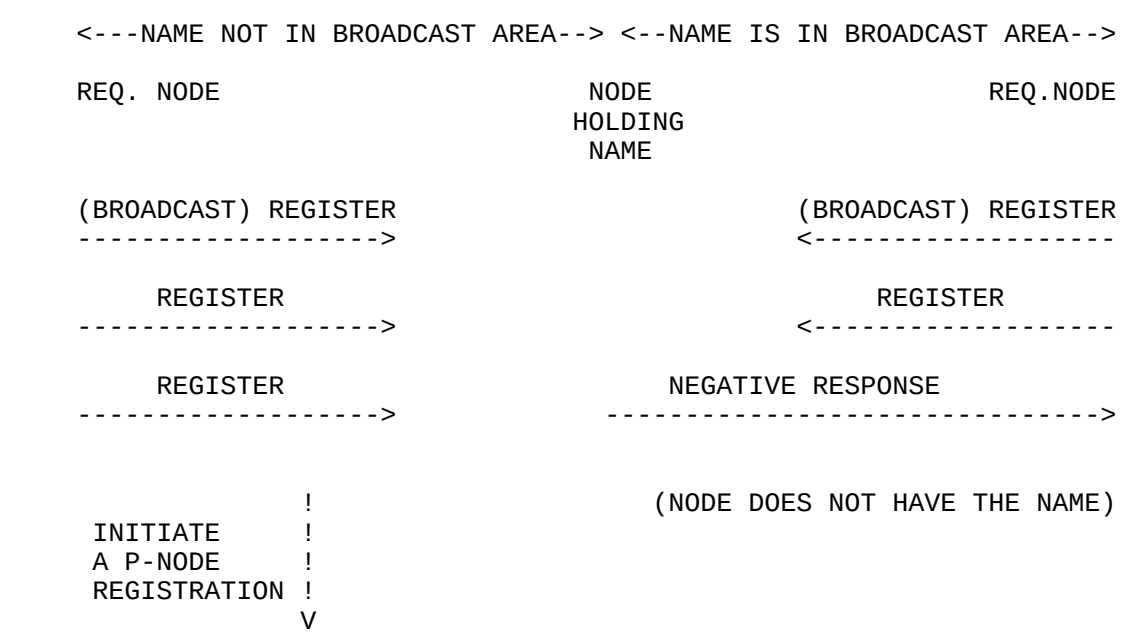
An M node begin a name claim operation as if the node were a B node: it broadcasts a NAME REGISTRATION REQUEST and listens for NEGATIVE NAME REGISTRATION RESPONSEs. Any NEGATIVE NAME REGISTRATION RESPONSE prevents the M node from obtaining the name and terminates the claim operation.

If, however, the M node does not receive any NEGATIVE NAME REGISTRATION RESPONSE, the M node must continue the claim procedure as if the M node were a P node.

Only if both name claims were successful does the M node acquire the name.

The following diagram illustrates M node name registration:

M-NODE REGISTRATION PROCESS



15.3. NAME QUERY TRANSACTIONS

Name query transactions are initiated by end-nodes to obtain the IP address(es) and other attributes associated with a NetBIOS name.

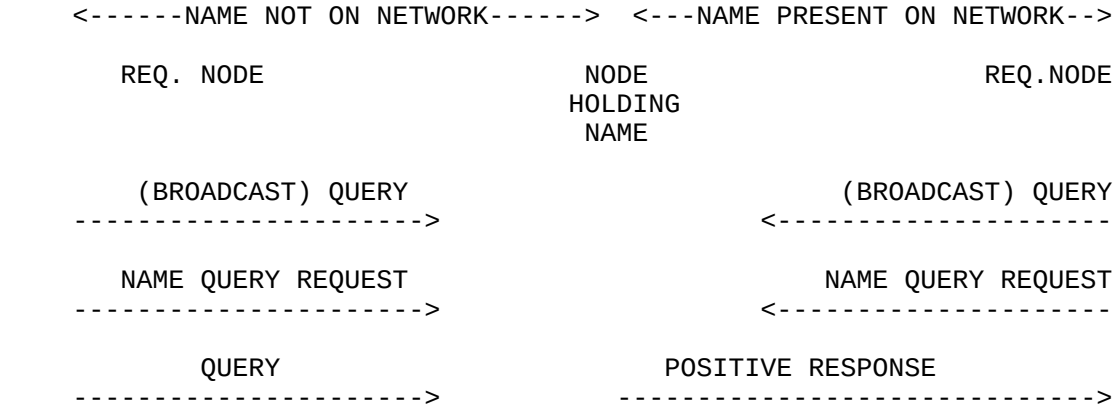
15.3.1. QUERY BY B NODES

The following diagram shows how B nodes go about discovering who owns a name.

The left half of the diagram illustrates what happens if there are no holders of the name. In that case no responses are received in answer to the broadcast NAME QUERY REQUEST(s).

The right half shows a POSITIVE NAME QUERY RESPONSE unicast by a name holder in answer to the broadcast request. A name holder will make this response to every NAME QUERY REQUEST that it hears. And each holder acts this way. Thus, the node sending the request may receive many responses, some duplicates, and from many nodes.

B-NODE DISCOVERY PROCESS



Name query is generally, but not necessarily, a prelude to NetBIOS session establishment or NetBIOS datagram transmission. However, name query may be used for other purposes.

A B node may elect to build a group membership list for subsequent use (e.g. for session establishment) by collecting and saving the responses.

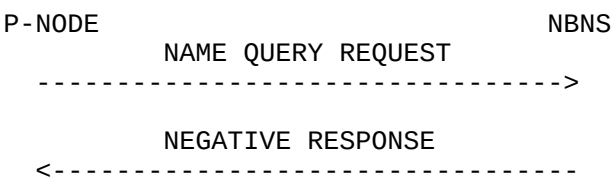
15.3.2. QUERY BY P NODES

An NBNS answers queries from a P node with a list of IP address and other information for each owner of the name. If there are multiple owners (i.e. if the name is a group name), the NBNS loads as many answers into the response as will fit into a UDP packet. A truncation flag indicates whether any additional owner information remains. All the information may be obtained by repeating the query over a TCP connection.

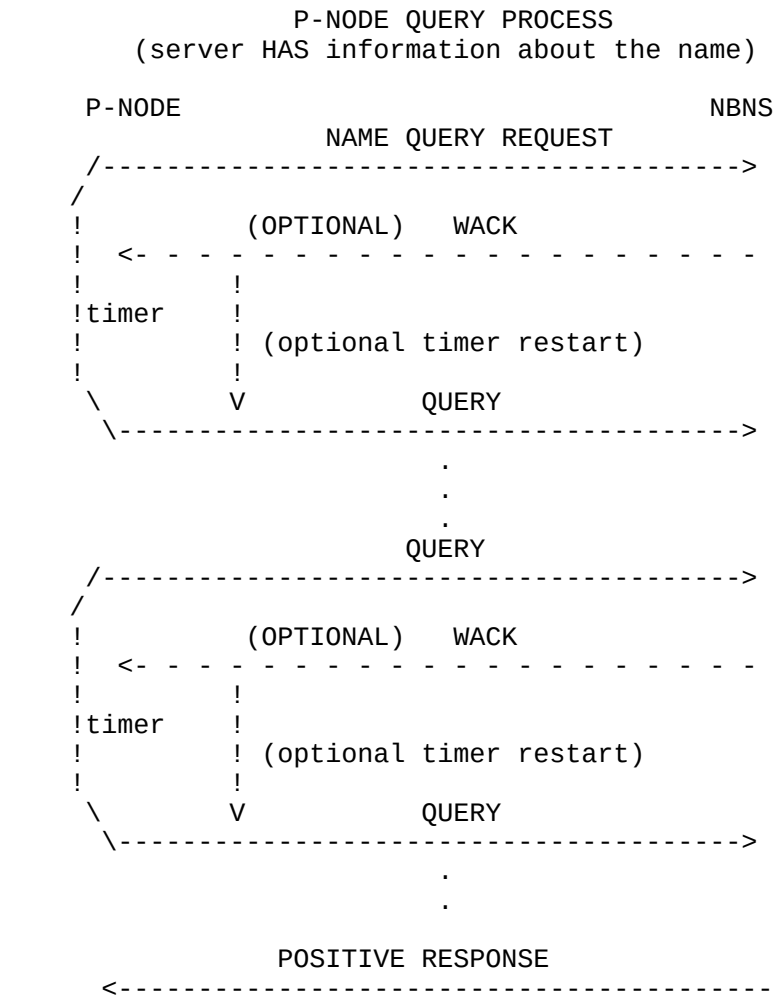
The NBNS is not required to impose any order on its answer list.

The following diagram shows what happens if the NBNS has no information about the name:

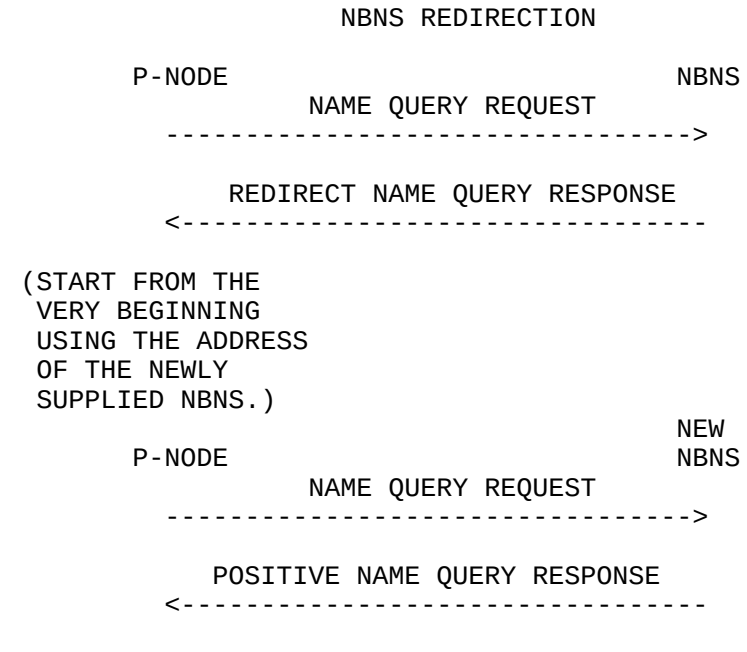
P-NODE DISCOVERY PROCESS
(server has no information about the name)



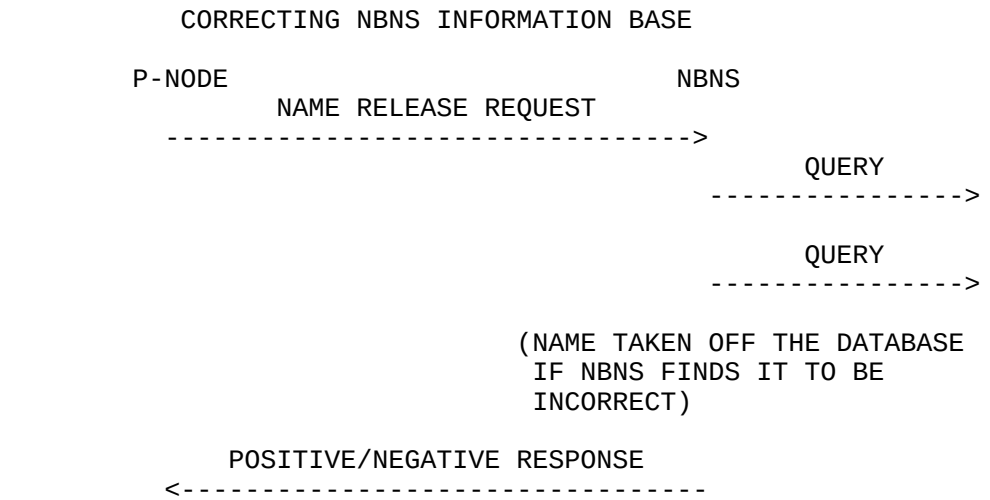
The next diagram illustrates interaction between the end-node and the NBNS when the NBNS does have information about the name. This diagram shows, in addition, the retransmission of the request by the end-node in the absence of a timely response. Also shown are WACKs (or temporary, intermediate responses) sent by the NBNS to the end-node:



The following diagram illustrates NBNS redirection. Upon receipt of a NAME QUERY REQUEST, the NBNS redirects the client to another NBNS. The client repeats the request to the new NBNS and obtains a response. The diagram shows that response as a POSITIVE NAME QUERY RESPONSE. However any legal NBNS response may occur in actual operation.



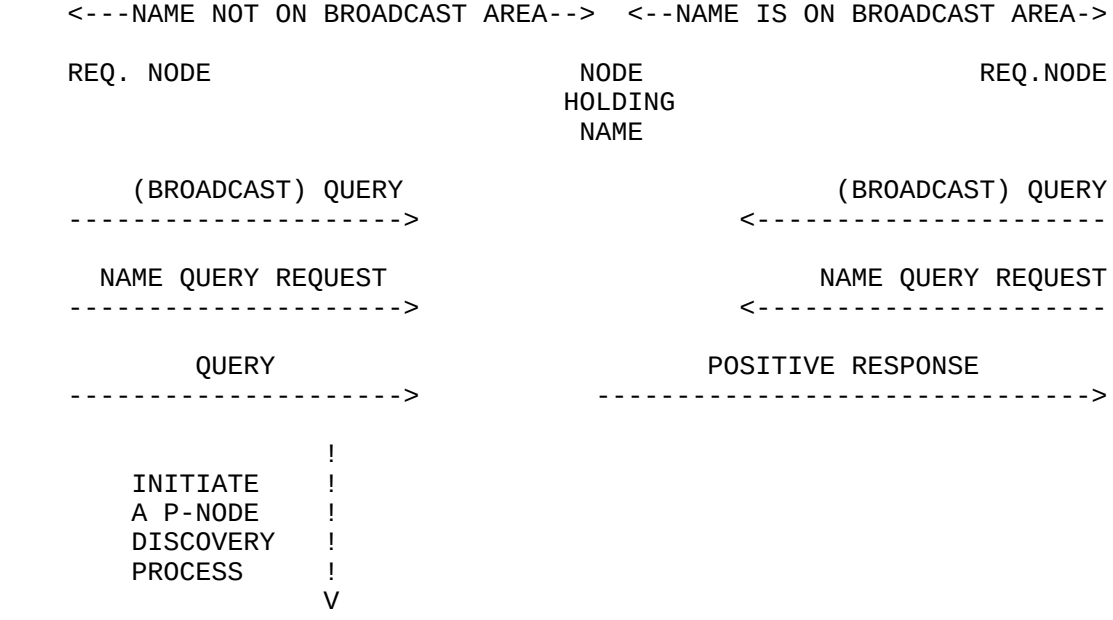
The next diagram shows how a P or M node tells the NBNS that the NBNS has provided incorrect information. This procedure may begin after a DATAGRAM ERROR packet has been received or a session set-up attempt has discovered that the NetBIOS name does not exist at the destination, the IP address of which was obtained from the NBNS during a prior name query transaction. The NBNS, in this case a secure NBNS, issues queries to verify whether the information is, in fact, incorrect. The NBNS closes the transaction by sending either a POSITIVE or NEGATIVE NAME RELEASE RESPONSE, depending on the results of the verification.



15.3.3. QUERY BY M NODES

M node name query follows the B node pattern. In the absence of adequate results, the M node then continues by performing a P node type query. This is shown in the following diagram:

M-NODE DISCOVERY PROCESS



15.3.4. ACQUIRE GROUP MEMBERSHIP LIST

The entire membership of a group may be acquired by sending a NAME QUERY REQUEST to the NBNS. The NBNS will respond with a POSITIVE NAME QUERY RESPONSE or a NEGATIVE NAME QUERY RESPONSE. A negative response completes the procedure and indicates that there are no members in the group.

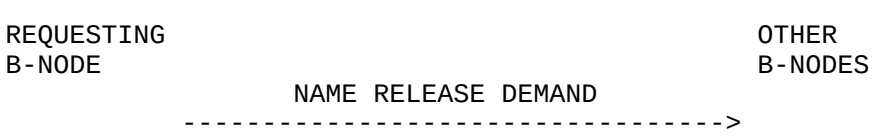
If the positive response has the truncation bit clear, then the response contains the entire list of group members. If the truncation bit is set, then this entire procedure must be repeated, but using TCP as a foundation rather than UDP.

15.4. NAME RELEASE TRANSACTIONS

15.4.1. RELEASE BY B NODES

A NAME RELEASE DEMAND contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID



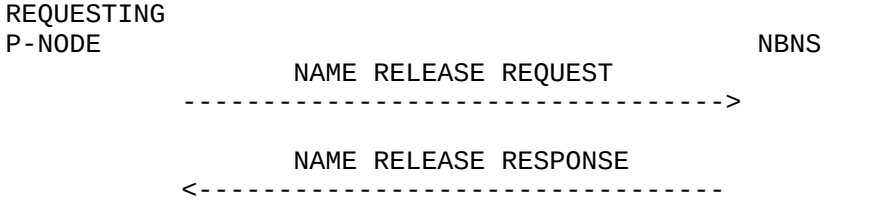
15.4.2. RELEASE BY P NODES

A NAME RELEASE REQUEST contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID

A NAME RELEASE RESPONSE contains the following information:

- NetBIOS name
- The scope of the NetBIOS name
- Name type: unique or group
- IP address of the releasing node
- Transaction ID
- Result:
  - Yes: name was released
  - No: name was not released, a reason code is provided



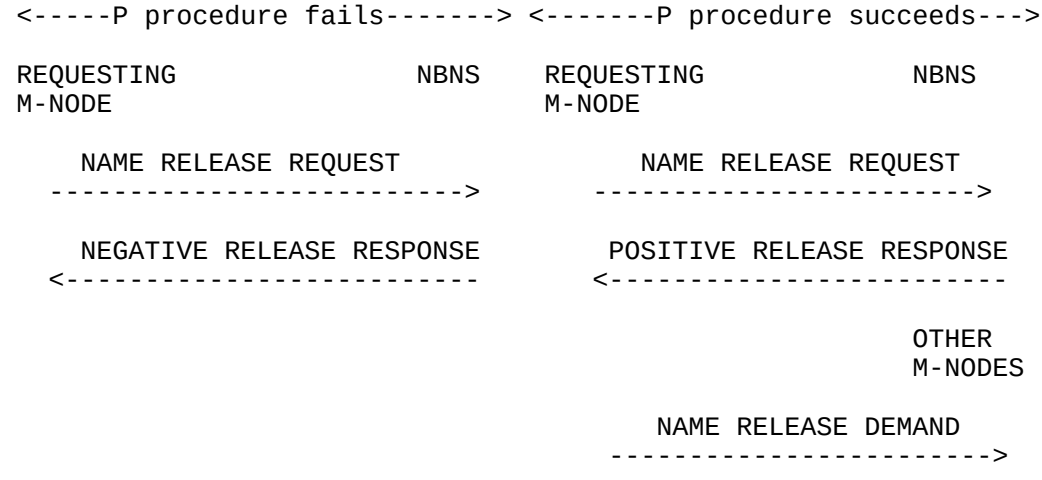
15.4.3. RELEASE BY M NODES

The name release procedure of the M node is a combination of the P and B node name release procedures. The M node first performs the P

release procedure. If the P procedure fails then the release procedure does not continue, it fails. If and only if the P procedure succeeds then the M node broadcasts the NAME RELEASE DEMAND to the broadcast area, the B procedure.

NOTE: An M node typically performs a B-style operation and then a P-style operation. In this case, however, the P-style operation comes first.

The following diagram illustrates the M node name release procedure:



15.5. NAME MAINTENANCE TRANSACTIONS

15.5.1. NAME REFRESH

Name refresh transactions are used to handle the following situations:

a) An NBNS node needs to detect if a P or M node has "silently" gone down, so that names held by that node can be purged from the data base.

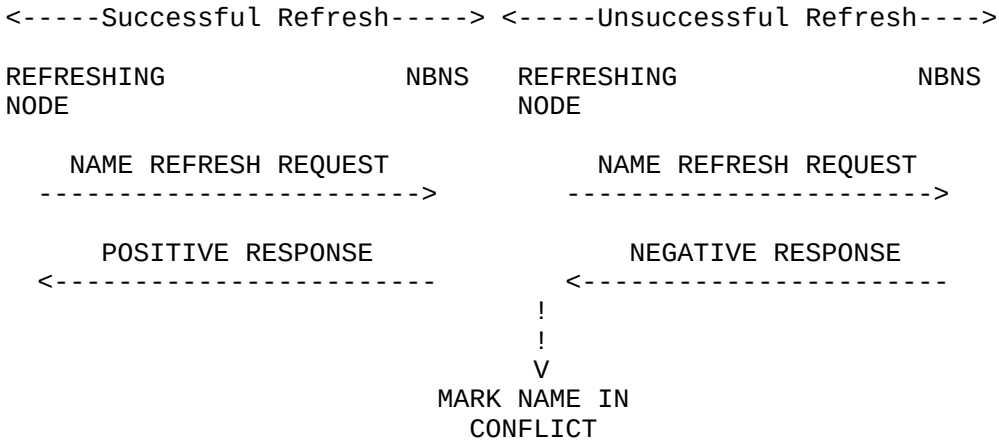
b) If the NBNS goes down, it needs to be able to reconstruct the data base when it comes back up.

c) If the network should be partitioned, the NBNS needs to be able to able to update its data base when the network reconnects.

Each P or M node is responsible for sending periodic NAME REFRESH REQUESTs for each name that it has registered. Each refresh packet contains a single name that has been successfully registered by that

node. The interval between such packets is negotiated between the end node and the NBNS server at the time that the name is initially claimed. At name claim time, an end node will suggest a refresh timeout value. The NBNS node can modify this value in the reply packet. A NBNS node can also choose to tell the end node to not send any refresh packet by using the "infinite" timeout value in the response packet. The timeout value returned by the NBNS is the actual refresh timeout that the end node must use.

When a node sends a NAME REFRESH REQUEST, it must be prepared to receive a negative response. This would happen, for example, if the the NBNS discovers that the the name had already been assigned to some other node. If such a response is received, the end node should mark the name as being in conflict. Such an entry should be treated in the same way as if name conflict had been detected against the name. The following diagram illustrates name refresh:



15.5.2. NAME CHALLENGE

Name challenge is done by sending a NAME QUERY REQUEST to an end node of any type. If a POSITIVE NAME QUERY RESPONSE is returned, then that node still owns the name. If a NEGATIVE NAME QUERY RESPONSE is received or if no response is received, it can be assumed that the end node no longer owns the name.

Name challenge can be performed either by the NBNS node, or by an end node. When an end-node sends a name claim packet, the NBNS node may do the challenge operation. The NBNS node can choose, however, to require the end node do the challenge. In that case, the NBNS will send an END-NODE CHALLENGE RESPONSE packet to the end node, which should then proceed to challenge the putative owner.

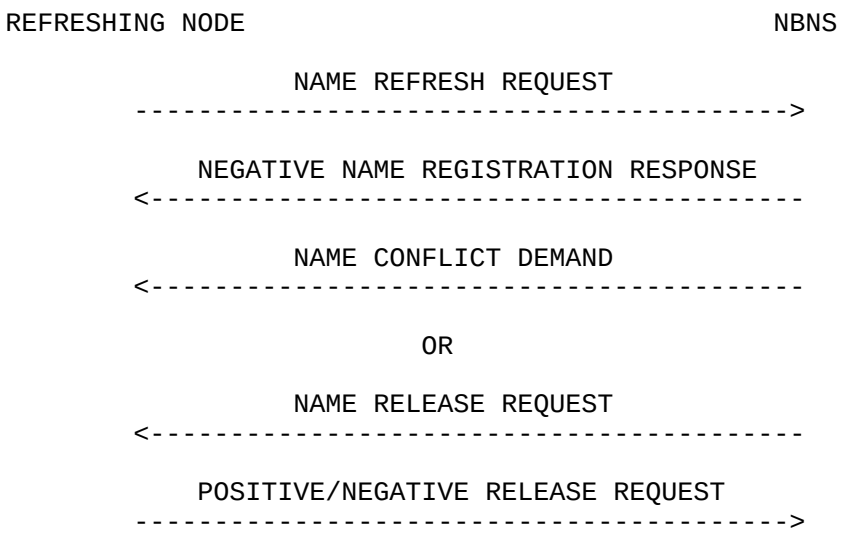
Note that the name challenge procedure sends a normal NAME QUERY REQUEST packet to the end node. It does not require a special packet. The only new packet introduced is the END-NODE CHALLENGE

RESPONSE which is sent by an NBNS node when the NBNS wants the end-node to perform the challenge operation.

15.5.3. CLEAR NAME CONFLICT

It is possible during a refresh request from a M or P node for a NBNS to detects a name in conflict. The response to the NAME REFRESH REQUEST must be a NEGATIVE NAME REGISTRATION RESPONSE. Optionally, in addition, the NBNS may send a NAME CONFLICT DEMAND or a NAME RELEASE REQUEST to the refreshing node. The NAME CONFLICT DEMAND forces the node to place the name in the conflict state. The node will eventually inform it's user of the conflict. The NAME RELEASE REQUEST will force the node to flush the name from its local name table completely. This forces the node to flush the name in conflict. This does not cause termination of existing sessions using this name.

The following diagram shows an NBNS detecting and correcting a conflict:



15.6. ADAPTER STATUS TRANSACTIONS

Adapter status is obtained from a node as follows:

1. Perform a name discovery operation to obtain the IP addresses of a set of end-nodes.

2. Repeat until all end-nodes from the set have been used:

   a. Select one end-node from the set.

   b. Send a NODE STATUS REQUEST to that end-node using UDP.

c. Await a NODE STATUS RESPONSE. (If a timely response is not forthcoming, repeat step "b" UCAST_REQ_RETRY_COUNT times. After the last retry, go to step "a".)

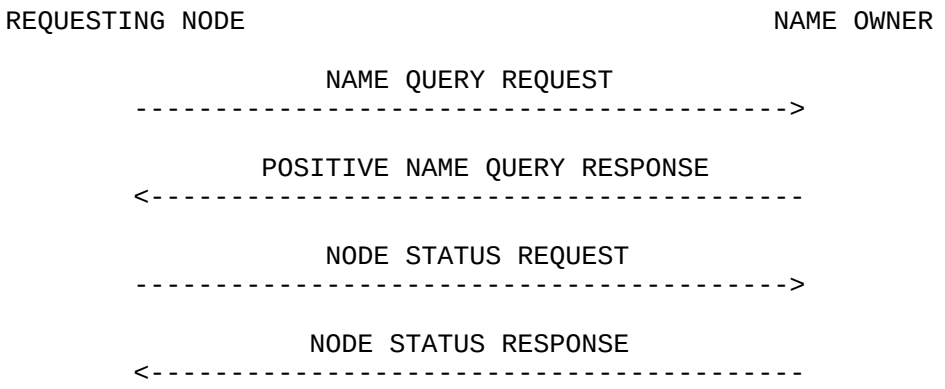
d. If the truncation bit is not set in the response, the response contains the entire node status. Return the status to the user and terminate this procedure.

e. If the truncation bit is set in the response, then not all status was returned because it would not fit into the response packet. The responder will set the truncation bit if the IP datagram length would exceed MAX_DATAGRAM_LENGTH. Return the status to the user and terminate this procedure.

3. Return error to user, no status obtained.

The repetition of step 2, above, through all nodes of the set, is optional.

Following is an example transaction of a successful Adapter Status operation:



16. NetBIOS SESSION SERVICE

The NetBIOS session service begins after one or more IP addresses have been found for the target name. These addresses may have been acquired using the NetBIOS name query transactions or by other means, such as a local name table or cache.

NetBIOS session service transactions, packets, and protocols are identical for all end-node types. They involve only directed (point-to-point) communications.

16.1. OVERVIEW OF NetBIOS SESSION SERVICE

Session service has three phases:

Session establishment - it is during this phase that the IP address and TCP port of the called name is determined, and a TCP connection is established with the remote party.

Steady state - it is during this phase that NetBIOS data messages are exchanged over the session. Keep-alive packets may also be exchanged if the participating nodes are so configured.

Session close - a session is closed whenever either a party (in the session) closes the session or it is determined that one of the parties has gone down.

16.1.1. SESSION ESTABLISHMENT PHASE OVERVIEW

An end-node begins establishment of a session to another node by somehow acquiring (perhaps using the name query transactions or a local cache) the IP address of the node or nodes purported to own the destination name.

Every end-node awaits incoming NetBIOS session requests by listening for TCP calls to a well-known service port, SSN_SRVC_TCP_PORT. Each incoming TCP connection represents the start of a separate NetBIOS session initiation attempt. The NetBIOS session server, not the ultimate application, accepts the incoming TCP connection(s).

Once the TCP connection is open, the calling node sends session service request packet. This packet contains the following information:

- Calling IP address (see note)
- Calling NetBIOS name
- Called IP address (see note)
- Called NetBIOS name

NOTE: The IP addresses are obtained from the TCP service interface.

When the session service request packet arrives at the NetBIOS server, one of the the following situations will exist:

- There exists a NetBIOS LISTEN compatible with the incoming call and there are adequate resources to permit session establishment to proceed.

- There exists a NetBIOS LISTEN compatible with the incoming call, but there are inadequate resources to permit

establishment of a session.

- The called name does, in fact, exist on the called node, but there is no pending NetBIOS LISTEN compatible with the incoming call.

- The called name does not exist on the called node.

In all but the first case, a rejection response is sent back over the TCP connection to the caller. The TCP connection is then closed and the session phase terminates. Any retry is the responsibility of the caller. For retries in the case of a group name, the caller may use the next member of the group rather than immediately retrying the instant address. In the case of a unique name, the caller may attempt an immediate retry using the same target IP address unless the called name did not exist on the called node. In that one case, the NetBIOS name should be re-resolved.

If a compatible LISTEN exists, and there are adequate resources, then the session server may transform the existing TCP connection into the NetBIOS data session. Alternatively, the session server may redirect, or "retarget" the caller to another TCP port (and IP address).

If the caller is redirected, the caller begins the session establishment anew, but using the new IP address and TCP port given in the retarget response. Again a TCP connection is created, and again the calling and called node exchange credentials. The called party may accept the call, reject the call, or make a further redirection.

This mechanism is based on the presumption that, on hosts where it is not possible to transfer open TCP connections between processes, the host will have a central session server. Applications willing to receive NetBIOS calls will obtain an ephemeral TCP port number, post a TCP unspecified passive open on that port, and then pass that port number and NetBIOS name information to the NetBIOS session server using a NetBIOS LISTEN operation. When the call is placed, the session server will "retarget" the caller to the application's TCP socket. The caller will then place a new call, directly to the application. The application has the responsibility to mimic the session server at least to the extent of receiving the calling credentials and then accepting or rejecting the call.

16.1.1.1. RETRYING AFTER BEING RETARGETTED

A calling node may find that it can not establish a session with a node to which it was directed by the retargetting procedure. Since retargetting may be nested, there is an issue whether the caller should begin a retry at the initial starting point or back-up to an intermediate retargetting point. The caller may use any method. A

discussion of two such methods is in Appendix B, "Retarget Algorithms".

16.1.1.2. SESSION ESTABLISHMENT TO A GROUP NAME

Session establishment with a group name requires special consideration. When a NetBIOS CALL attempt is made to a group name, name discovery will result in a list (possibly incomplete) of the members of that group. The calling node selects one member from the list and attempts to build a session. If that fails, the calling node may select another member and make another attempt.

When the session service attempts to make a connection with one of the members of the group, there is no guarantee that that member has a LISTEN pending against that group name, that the called node even owns, or even that the called node is operating.

16.1.2. STEADY STATE PHASE OVERVIEW

NetBIOS data messages are exchanged in the steady state. NetBIOS messages are sent by prepending the user data with a message header and sending the header and the user data over the TCP connection. The receiver removes the header and passes the data to the NetBIOS user.

In order to detect failure of one of the nodes or of the intervening network, "session keep alive" packets may be periodically sent in the steady state.

Any failure of the underlying TCP connection, whether a reset, a timeout, or other failure, implies failure of the NetBIOS session.

16.1.3. SESSION TERMINATION PHASE OVERVIEW

A NetBIOS session is terminated normally when the user requests the session to be closed or when the session service detects the remote partner of the session has gracefully terminated the TCP connection. A NetBIOS session is abnormally terminated when the session service detects a loss of the connection. Connection loss can be detected with the keep-alive function of the session service or TCP, or on the failure of a SESSION MESSAGE send operation.

When a user requests to close a session, the service first attempts a graceful in-band close of the TCP connection. If the connection does not close within the SSN_CLOSE_TIMEOUT the TCP connection is aborted. No matter how the TCP connection is terminated, the NetBIOS session service always closes the NetBIOS session.

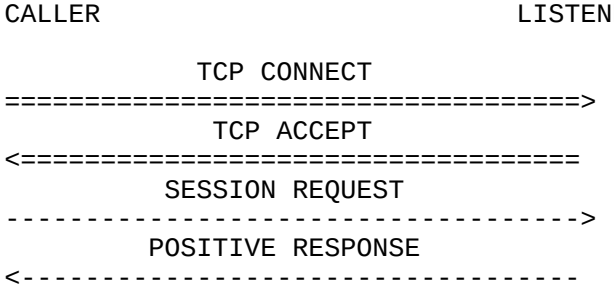
When the session service receives an indication from TCP that a connection close request has been received, the TCP connection and the NetBIOS session are immediately closed and the user is informed

of the loss of the session. All data received up to the close indication should be delivered, if possible, to the session's user.

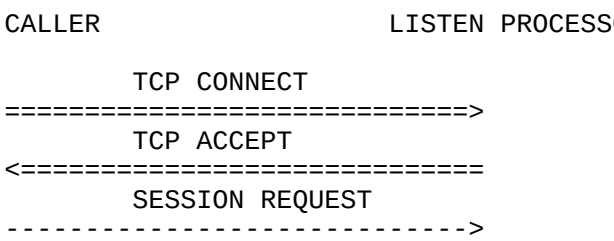
16.2. SESSION ESTABLISHMENT PHASE

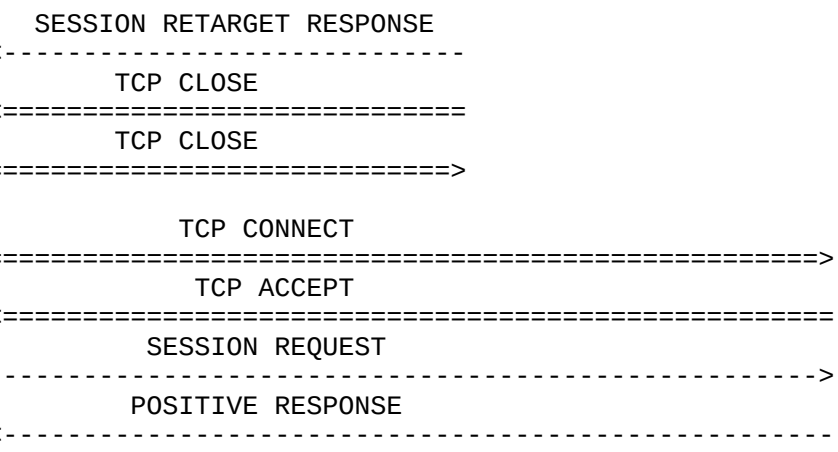
All the following diagrams assume a name query operation was successfully completed by the caller node for the listener's name.

This first diagram shows the sequence of network events used to successfully establish a session without retargetting by the listener. The TCP connection is first established with the well-known NetBIOS session service TCP port, SSN_SRVC_TCP_PORT. The caller then sends a SESSION REQUEST packet over the TCP connection requesting a session with the listener. The SESSION REQUEST contains the caller's name and the listener's name. The listener responds with a POSITIVE SESSION RESPONSE informing the caller this TCP connection is accepted as the connection for the data transfer phase of the session.

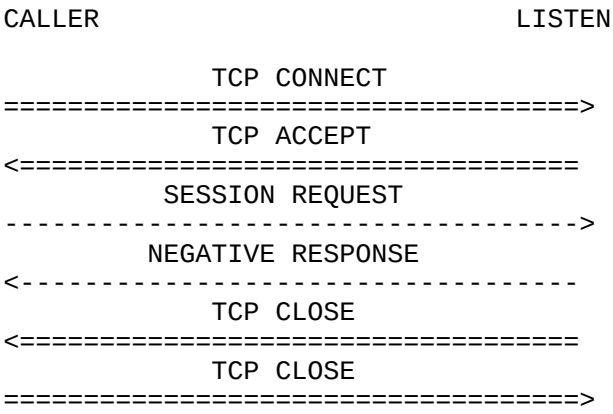


The second diagram shows the sequence of network events used to successfully establish a session when the listener does retargetting. The session establishment procedure is the same as with the first diagram up to the listener's response to the SESSION REQUEST. The listener, divided into two sections, the listen processor and the actual listener, sends a SESSION RETARGET RESPONSE to the caller. This response states the call is acceptable, but the data transfer TCP connection must be at the new IP address and TCP port. The caller then re-iterates the session establishment process anew with the new IP address and TCP port after the initial TCP connection is closed. The new listener then accepts this connection for the data transfer phase with a POSITIVE SESSION RESPONSE.

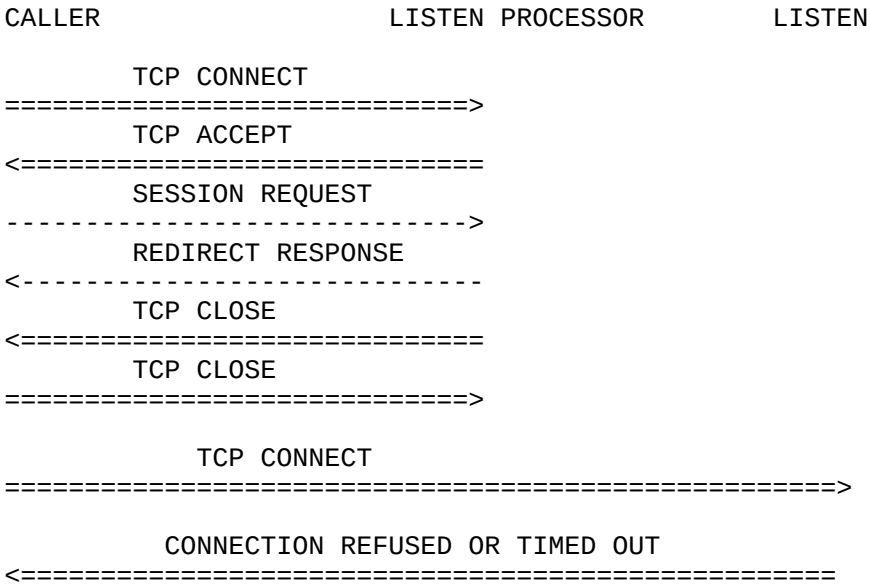




The third diagram is the sequence of network events for a rejected session request with the listener. This type of rejection could occur with either a non-retargetting listener or a retargetting listener. After the TCP connection is established at SSN_SRVC_TCP_PORT, the caller sends the SESSION REQUEST over the TCP connection. The listener does not have either a listen pending for the listener's name or the pending NetBIOS listen is specific to another caller's name. Consequently, the listener sends a NEGATIVE SESSION RESPONSE and closes the TCP connection.



The fourth diagram is the sequence of network events when session establishment fails with a retargetting listener. After being redirected, and after the initial TCP connection is closed the caller tries to establish a TCP connection with the new IP address and TCP port. The connection fails because either the port is unavailable or the target node is not active. The port unavailable race condition occurs if another caller has already acquired the TCP connection with the listener. For additional implementation suggestions, see Appendix B, "Retarget Algorithms".



16.3. SESSION DATA TRANSFER PHASE

16.3.1. DATA ENCAPSULATION

NetBIOS messages are exchanged in the steady state. Messages are sent by prepending user data with message header and sending the header and the user data over the TCP connection. The receiver removes the header and delivers the NetBIOS data to the user.

16.3.2. SESSION KEEP-ALIVES

In order to detect node failure or network partitioning, "session keep alive" packets are periodically sent in the steady state. A session keep alive packet is discarded by a peer node.

A session keep alive timer is maintained for each session. This timer is reset whenever any data is sent to, or received from, the session peer. When the timer expires, a NetBIOS session keep-alive packet is sent on the TCP connection. Sending the keep-alive packet forces data to flow on the TCP connection, thus indirectly causing TCP to detect whether the connection is still active.

Since many TCP implementations provide a parallel TCP "keep- alive" mechanism, the NetBIOS session keep-alive is made a configurable option. It is recommended that the NetBIOS keep- alive mechanism be used only in the absence of TCP keep-alive.

Note that unlike TCP keep alives, NetBIOS session keep alives do not require a response from the NetBIOS peer -- the fact that it was

possible to send the NetBIOS session keep alive is sufficient indication that the peer, and the connection to it, are still active.

The only requirement for interoperability is that when a session keep alive packet is received, it should be discarded.

17. NETBIOS DATAGRAM SERVICE

17.1. OVERVIEW OF NetBIOS DATAGRAM SERVICE

Every NetBIOS datagram has a named destination and source. To transmit a NetBIOS datagram, the datagram service must perform a name query operation to learn the IP address and the attributes of the destination NetBIOS name. (This information may be cached to avoid the overhead of name query on subsequent NetBIOS datagrams.)

NetBIOS datagrams are carried within UDP packets. If a NetBIOS datagram is larger than a single UDP packet, it may be fragmented into several UDP packets.

End-nodes may receive NetBIOS datagrams addressed to names not held by the receiving node. Such datagrams should be discarded. If the name is unique then a DATAGRAM ERROR packet is sent to the source of that NetBIOS datagram.

17.1.1. UNICAST, MULTICAST, AND BROADCAST

NetBIOS datagrams may be unicast, multicast, or broadcast. A NetBIOS datagram addressed to a unique NetBIOS name is unicast. A NetBIOS datatgram addressed to a group NetBIOS name, whether there are zero, one, or more actual members, is multicast. A NetBIOS datagram sent using the NetBIOS "Send Broadcast Datagram" primitive is broadcast.

17.1.2. FRAGMENTATION OF NetBIOS DATAGRAMS

When the header and data of a NetBIOS datagram exceeds the maximum amount of data allowed in a UDP packet, the NetBIOS datagram must be fragmented before transmission and reassembled upon receipt.

A NetBIOS Datagram is composed of the following protocol elements:

- IP header of 20 bytes (minimum)
- UDP header of 8 bytes
- NetBIOS Datagram Header of 14 bytes
- The NetBIOS Datagram data.

The NetBIOS Datagram data section is composed of 3 parts:

- Source NetBIOS name (255 bytes maximum)
- Destination NetBIOS name (255 bytes maximum)
- The NetBIOS user’s data (maximum of 512 bytes)

The two name fields are in second level encoded format (see section 14.)

A maximum size NetBIOS datagram is 1064 bytes. The minimal maximum IP datagram size is 576 bytes. Consequently, a NetBIOS Datagram may not fit into a single IP datagram. This makes it necessary to permit the fragmentation of NetBIOS Datagrams.

On networks meeting or exceeding the minimum IP datagram length requirement of 576 octets, at most two NetBIOS datagram fragments will be generated. The protocols and packet formats accommodate fragmentation into three or more parts.

When a NetBIOS datagram is fragmented, the IP, UDP and NetBIOS Datagram headers are present in each fragment. The NetBIOS Datagram data section is split among resulting UDP datagrams. The data sections of NetBIOS datagram fragments do not overlap. The only fields of the NetBIOS Datagram header that would vary are the FLAGS and OFFSET fields.

The FIRST bit in the FLAGS field indicate whether the fragment is the first in a sequence of fragments. The MORE bit in the FLAGS field indicates whether other fragments follow.

The OFFSET field is the byte offset from the beginning of the NetBIOS datagram data section to the first byte of the data section in a fragment. It is 0 for the first fragment. For each subsequent fragment, OFFSET is the sum of the bytes in the NetBIOS data sections of all preceding fragments.

If the NetBIOS datagram was not fragmented:

- FIRST = TRUE
- MORE = FALSE
- OFFSET = 0

If the NetBIOS datagram was fragmented:

- First fragment:
  - FIRST = TRUE
  - MORE = TRUE
  - OFFSET = 0
- Intermediate fragments:
  - FIRST = FALSE
  - MORE = TRUE
  - OFFSET = sum(NetBIOS data in prior fragments)
- Last fragment:
  - FIRST = FALSE
  - MORE = FALSE

- OFFSET = sum(NetBIOS data in prior fragments)

The relative position of intermediate fragments may be ascertained from OFFSET.

An NBDD must remember the destination name of the first fragment in order to relay the subsequent fragments of a single NetBIOS datagram. The name information can be associated with the subsequent fragments through the transaction ID, DGM_ID, and the SOURCE_IP, fields of the packet. This information can be purged by the NBDD after the last fragment has been processed or FRAGMENT_TO time has expired since the first fragment was received.

17.2. NetBIOS DATAGRAMS BY B NODES

For NetBIOS datagrams with a named destination (i.e. non- broadcast), a B node performs a name discovery for the destination name before sending the datagram. (Name discovery may be bypassed if information from a previous discovery is held in a cache.) If the name type returned by name discovery is UNIQUE, the datagram is unicast to the sole owner of the name. If the name type is GROUP, the datagram is broadcast to the entire broadcast area using the destination IP address BROADCAST_ADDRESS.

A receiving node always filters datagrams based on the destination name. If the destination name is not owned by the node or if no RECEIVE DATAGRAM user operations are pending for the name, then the datagram is discarded. For datagrams with a UNIQUE name destination, if the name is not owned by the node then the receiving node sends a DATAGRAM ERROR packet. The error packet originates from the DGM_SRVC_UDP_PORT and is addressed to the SOURCE_IP and SOURCE_PORT from the bad datagram. The receiving node quietly discards datagrams with a GROUP name destination if the name is not owned by the node.

Since broadcast NetBIOS datagrams do not have a named destination, the B node sends the DATAGRAM SERVICE packet(s) to the entire broadcast area using the destination IP address BROADCAST_ADDRESS. In order for the receiving nodes to distinguish this datagram as a broadcast NetBIOS datagram, the NetBIOS name used as the destination name is '*' (hexadecimal 2A) followed by 15 bytes of hexidecimal 00. The NetBIOS scope identifier is appended to the name before it is converted into second-level encoding. For example, if the scope identifier is "NETBIOS.SCOPE" then the first-level encoded name would be:

CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.NETBIOS.SCOPE

According to [2], a user may not provide a NetBIOS name beginning with "*".

For each node in the broadcast area that receives the NetBIOS

broadcast datagram, if any RECEIVE BROADCAST DATAGRAM user operations are pending then the data from the NetBIOS datagram is replicated and delivered to each. If no such operations are pending then the node silently discards the datagram.

17.3. NetBIOS DATAGRAMS BY P AND M NODES

P and M nodes do not use IP broadcast to distribute NetBIOS datagrams.

Like B nodes, P and M nodes must perform a name discovery or use cached information to learn whether a destination name is a group or a unique name.

Datagrams to unique names are unicast directly to the destination by P and M nodes, exactly as they are by B nodes.

Datagrams to group names and NetBIOS broadcast datagrams are unicast to the NBDD. The NBDD then relays the datagrams to each of the nodes specified by the destination name.

An NBDD may not be capable of sending a NetBIOS datagram to a particular NetBIOS name, including the broadcast NetBIOS name ("*") defined above. A query mechanism is available to the end- node to determine if a NBDD will be able to relay a datagram to a given name. Before a datagram, or its fragments, are sent to the NBDD the P or M node may send a DATAGRAM QUERY REQUEST packet to the NBDD with the DESTINATION_NAME from the DATAGRAM SERVICE packet(s). The NBDD will respond with a DATAGRAM POSITIVE QUERY RESPONSE if it will relay datagrams to the specified destination name. After a positive response the end-node unicasts the datagram to the NBDD. If the NBDD will not be able to relay a datagram to the destination name specified in the query, a DATAGRAM NEGATIVE QUERY RESPONSE packet is returned. If the NBDD can not distribute a datagram, the end-node then has the option of getting the name's owner list from the NBNS and sending the datagram directly to each of the owners.

An NBDD must be able to respond to DATAGRAM QUERY REQUEST packets. The response may always be positive. However, the usage or implementation of the query mechanism by a P or M node is optional. An implementation may always unicast the NetBIOS datagram to the NBDD without asking if it will be relayed. Except for the datagram query facility described above, an NBDD provides no feedback to indicate whether it forwarded a datagram.

18. NODE CONFIGURATION PARAMETERS

- B NODES:
  - Node's permanent unique name
  - Whether IGMP is in use
  - Broadcast IP address to use

        - Whether NetBIOS session keep-alives are needed
        - Usable UDP data field length (to control fragmentation)
    - P NODES:
        - Node's permanent unique name
        - IP address of NBNS
        - IP address of NBDD
        - Whether NetBIOS session keep-alives are needed
        - Usable UDP data field length (to control fragmentation)
    - M NODES:
        - Node's permanent unique name
        - Whether IGMP is in use
        - Broadcast IP address to use
        - IP address of NBNS
        - IP address of NBDD
        - Whether NetBIOS session keep-alives are needed
        - Usable UDP data field length (to control fragmentation)

19. MINIMAL CONFORMANCE

To ensure multi-vendor interoperability, a minimally conforming implementation based on this specification must observe the following rules:

a) A node designed to work only in a broadcast area must conform to the B node specification.

b) A node designed to work only in an internet must conform to the P node specification.

REFERENCES

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987.

[2] IBM Corp., "IBM PC Network Technical Reference Manual", No. 6322916, First Edition, September 1984.

[3] J. Postel (Ed.), "Transmission Control Protocol", RFC 793, September 1981.

[4] MIL-STD-1778

[5] J. Postel, "User Datagram Protocol", RFC 768, 28 August 1980.

[6] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[7] J. Postel, "Internet Protocol", RFC 791, September 1981.

[8] J. Mogul, "Internet Subnets", RFC 950, October 1984

[9] J. Mogul, "Broadcasting Internet Datagrams in the Presence of Subnets", RFC 922, October 1984.

[10] J. Mogul, "Broadcasting Internet Datagrams", RFC 919, October 1984.

[11] P. Mockapetris, "Domain Names - Concepts and Facilities", RFC 882, November 1983.

[12] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.

[13] P. Mockapetris, "Domain System Changes and Observations", RFC 973, January 1986.

[14] C. Partridge, "Mail Routing and the Domain System", RFC 974, January 1986.

[15] S. Deering, D. Cheriton, "Host Groups: A Multicast Extension to the Internet Protocol", RFC 966, December 1985.

[16] S. Deering, "Host Extensions for IP Multicasting", RFC 988, July 1986.

APPENDIX A

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

INTEGRATION WITH INTERNET GROUP MULTICASTING

The Netbios-over-TCP system described in this RFC may be easily integrated with the Internet Group Multicast system now being developed for the internet.

In the main body of the RFC, the notion of a broadcast area was considered to be a single MAC-bridged "B-LAN". However, the protocols defined will operate over an extended broadcast area resulting from the creation of a permanent Internet Multicast Group.

Each separate broadcast area would be based on a separate permanent Internet Multicast Group. This multicast group address would be used by B and M nodes as their BROADCAST_ADDRESS.

In order to base the broadcast area on a multicast group certain additional procedures are required and certain constraints must be met.

A-1. ADDITIONAL PROTOCOL REQUIRED IN B AND M NODES

All B or M nodes operating on an IGMP based broadcast area must have IGMP support in their IP layer software. These nodes must perform an IGMP join operation to enter the IGMP group before engaging in NetBIOS activity.

A-2. CONSTRAINTS

Broadcast Areas may overlap. For this reason, end-nodes must be careful to examine the NetBIOS scope identifiers in all received broadcast packets.

The NetBIOS broadcast protocols were designed for a network that exhibits a low average transit time and low rate of packet loss. An IGMP based broadcast area must exhibit these characteristics. In practice this will tend to constrain IGMP broadcast areas to a campus of networks interconnected by high-speed routers and inter-router links. It is unlikely that transcontinental broadcast areas would exhibit the required characteristics.

APPENDIX B

This appendix contains supporting technical discussions. It is not an integral part of the NetBIOS-over-TCP specification.

IMPLEMENTATION CONSIDERATIONS

B-1. IMPLEMENTATION MODELS

On any participating system, there must be some sort of NetBIOS Service to coordinate access by NetBIOS applications on that system.

To analyze the impact of the NetBIOS-over-TCP architecture, we use the following three models of how a NetBIOS service might be implemented:

1. Combined Service and Application Model

   The NetBIOS service and application are both contained within a single process. No interprocess communication is assumed within the system; all communication is over the network. If multiple applications require concurrent access to the NetBIOS service, they must be folded into this monolithic process.

2. Common Kernel Element Model

   The NetBIOS Service is part of the operating system (perhaps as a device driver or a front-end processor). The NetBIOS applications are normal operating system application processes. The common element NetBIOS service contains all the information, such as the name and listen tables, required to co-ordinate the activities of the applications.

3. Non-Kernel Common Element Model

   The NetBIOS Service is implemented as an operating system application process. The NetBIOS applications are other operating system application processes. The service and the applications exchange data via operating system interprocess communication. In a multi-processor (e.g. network) operating system, each module may reside on a different cpu. The NetBIOS service process contains all the shared information required to coordinate the activities of the NetBIOS applications. The applications may still require a subroutine library to facilitate access to the NetBIOS service.

For any of the implementation models, the TCP/IP service can be located in the operating system or split among the NetBIOS applications and the NetBIOS service processes.

B-1.1 MODEL INDEPENDENT CONSIDERATIONS

The NetBIOS name service associates a NetBIOS name with a host. The NetBIOS session service further binds the name to a specific TCP port for the duration of the session.

The name service does not need to be informed of every Listen initiation and completion. Since the names are not bound to any TCP port in the name service, the session service may use a different tcp port for each session established with the same local name.

The TCP port used for the data transfer phase of a NetBIOS session can be globally well-known, locally well-known, or ephemeral. The choice is a local implementation issue. The RETARGET mechanism allows the binding of the NetBIOS session to a TCP connection to any TCP port, even to another IP node.

An implementation may use the session service's globally well- known TCP port for the data transfer phase of the session by not using the RETARGET mechanism and, rather, accepting the session on the initial TCP connection. This is permissible because the caller always uses an ephemeral TCP port.

The complexity of the called end RETARGET mechanism is only required if a particular implementation needs it. For many real system environments, such as an in-kernel NetBIOS service implementation, it will not be necessary to retarget incoming calls. Rather, all NetBIOS sessions may be multiplexed through the single, well-known, NetBIOS session service port. These implementations will not be burdened by the complexity of the RETARGET mechanism, nor will their callers be required to jump through the retargetting hoops.

Nevertheless, all callers must be ready to process all possible SESSION RETARGET RESPONSEs.

B-1.2 SERVICE OPERATION FOR EACH MODEL

It is possible to construct a NetBIOS service based on this specification for each of the above defined implementation models.

For the common kernel element model, all the NetBIOS services, name, datagram, and session, are simple. All the information is contained within a single entity and can therefore be accessed or modified easily. A single port or multiple ports for the NetBIOS sessions can be used without adding any significant complexity to the session establishment procedure. The only penalty is the amount of overhead incurred to get the NetBIOS messages and operation requests/responses

through the user and operating system boundary.

The combined service and application model is very similar to the common kernel element model in terms of its requirements on the NetBIOS service. The major difficulty is the internal coordination of the multiple NetBIOS service and application processes existing in a system of this type.

The NetBIOS name, datagram and session protocols assume that the entities at the end-points have full control of the various well-known TCP and UDP ports. If an implementation has multiple NetBIOS service entities, as would be the case with, for example, multiple applications each linked into a NetBIOS library, then that implementation must impose some internal coordination. Alternatively, use of the NetBIOS ports could be periodically assigned to one application or another.

For the typical non-kernel common element mode implementation, three permanent system-wide NetBIOS service processes would exist:

- The name server
- the datagram server
- and session server

Each server would listen for requests from the network on a UDP or TCP well-known port. Each application would have a small piece of the NetBIOS service built-in, possibly a library. Each application's NetBIOS support library would need to send a message to the particular server to request an operation, such as add name or send a datagram or set-up a listen.

The non-kernel common element model does not require a TCP connection be passed between the two processes, session server and application. The RETARGET operation for an active NetBIOS Listen could be used by the session server to redirect the session to another TCP connection on a port allocated and owned by the application's NetBIOS support library. The application with either a built-in or a kernel-based TCP/IP service could then accept the RETARGETed connection request and process it independently of the session server.

On Unix(tm) or POSIX(tm), the NetBIOS session server could create sub-processes for incoming connections. The open sessions would be passed through "fork" and "exec" to the child as an open file descriptor. This approach is very limited, however. A pre- existing process could not receive an incoming call. And all call-ed processes would have to be sub-processes of the session server.

B-2. CASUAL AND RESTRICTED NetBIOS APPLICATIONS

Because NetBIOS was designed to operate in the open system environment of the typical personal computer, it does not have the

concept of privileged or unprivileged applications. In many multi-user or multi-tasking operating systems applications are assigned privilege capabilities. These capabilities limit the applications ability to acquire and use system resources. For these systems it is important to allow casual applications, those with limited system privileges, and privileged applications, those with 'super-user' capabilities but access to them and their required resources is restricted, to access NetBIOS services. It is also important to allow a systems administrator to restrict certain NetBIOS resources to a particular NetBIOS application. For example, a file server based on the NetBIOS services should be able to have names and TCP ports for sessions only it can use.

A NetBIOS application needs at least two local resources to communicate with another NetBIOS application, a NetBIOS name for itself and, typically, a session. A NetBIOS service cannot require that NetBIOS applications directly use privileged system resources. For example, many systems require privilege to use TCP and UDP ports with numbers less than 1024. This RFC requires reserved ports for the name and session servers of a NetBIOS service implementation. It does not require the application to have direct access these reserved ports.

For the name service, the manager of the local name table must have access to the NetBIOS name service's reserved UDP port. It needs to listen for name service UDP packets to defend and define its local names to the network. However, this manager need not be a part of a user application in a system environment which has privilege restrictions on reserved ports.

The internal name server can require certain privileges to add, delete, or use a certain name, if an implementer wants the restriction. This restriction is independent of the operation of the NetBIOS service protocols and would not necessarily prevent the interoperation of that implementation with another implementation.

The session server is required to own a reserved TCP port for session establishment. However, the ultimate TCP connection used to transmit and receive data does not have to be through that reserved port. The RETARGET procedure the NetBIOS session to be shifted to another TCP connection, possibly through a different port at the called end. This port can be an unprivileged resource, with a value greater than 1023. This facilitates casual applications.

Alternately, the RETARGET mechanism allows the TCP port used for data transmission and reception to be a reserved port. Consequently, an application wishing to have access to its ports maintained by the system administrator can use these restricted TCP ports. This facilitates privileged applications.

A particular implementation may wish to require further special

privileges for session establishment, these could be associated with internal information. It does not have to be based solely on TCP port allocation. For example, a given NetBIOS name may only be used for sessions by applications with a certain system privilege level.

The decision to use reserved or unreserved ports or add any additional name registration and usage authorization is a purely local implementation decision. It is not required by the NetBIOS protocols specified in the RFC.

B-3. TCP VERSUS SESSION KEEP-ALIVES

The KEEP-ALIVE is a protocol element used to validate the existence of a connection. A packet is sent to the remote connection partner to solicit a response which shows the connection is still functioning. TCP KEEP-ALIVES are used at the TCP level on TCP connections while session KEEP-ALIVES are used on NetBIOS sessions. These protocol operations are always transparent to the connection user. The user will only find out about a KEEP-ALIVE operation if it fails, therefore, if the connection is lost.

The NetBIOS specification[2] requires the NetBIOS service to inform the session user if a session is lost when it is in a passive or active state. Therefore,if the session user is only waiting for a receive operation and the session is dropped the NetBIOS service must inform the session user. It cannot wait for a session send operation before it informs the user of the loss of the connection.

This requirement stems from the management of scarce or volatile resources by a NetBIOS application. If a particular user terminates a session with a server application by destroying the client application or the NetBIOS service without a NetBIOS Hang Up, the server application will want to clean-up or free allocated resources. This server application if it only receives and then sends a response requires the notification of the session abort in the passive state.

The standard definition of a TCP service cannot detect loss of a connection when it is in a passive state, waiting for a packet to arrive. Some TCP implementations have added a KEEP-ALIVE operation which is interoperable with implementations without this feature. These implementations send a packet with an invalid sequence number to the connection partner. The partner, by specification, must respond with a packet showing the correct sequence number of the connection. If no response is received from the remote partner within a certain time interval then the TCP service assumes the connection is lost.

Since many TCP implementations do not have this KEEP-ALIVE function an optional NetBIOS KEEP-ALIVE operation has been added to the NetBIOS session protocols. The NetBIOS KEEP-ALIVE uses the properties of TCP to solicit a response from the remote connection

partner. A NetBIOS session message called KEEP-ALIVE is sent to the remote partner. Since this results in TCP sending an IP packet to the remote partner, the TCP connection is active. TCP will discover if the TCP connection is lost if the remote TCP partner does not acknowledge the IP packet. Therefore, the NetBIOS session service does not send a response to a session KEEP ALIVE message. It just throws it away. The NetBIOS session service that transmits the KEEP ALIVE is informed only of the failure of the TCP connection. It does not wait for a specific response message.

A particular NetBIOS implementation should use KEEP-ALIVES if it is concerned with maintaining compatibility with the NetBIOS interface specification[2]. Compatibility is especially important if the implementation wishes to support existing NetBIOS applications, which typically require the session loss detection on their servers, or future applications which were developed for implementations with session loss detection.

B-4. RETARGET ALGORITHMS

This section contains 2 suggestions for RETARGET algorithms. They are called the "straight" and "stack" methods. The algorithm in the body of the RFC uses the straight method. Implementation of either algorithm must take into account the Session establishment maximum retry count. The retry count is the maximum number of TCP connect operations allowed before a failure is reported.

The straight method forces the session establishment procedure to begin a retry after a retargetting failure with the initial node returned from the name discovery procedure. A retargetting failure is when a TCP connection attempt fails because of a time- out or a NEGATIVE SESSION RESPONSE is received with an error code specifying NOT LISTENING ON CALLED NAME. If any other failure occurs the session establishment procedure should retry from the call to the name discovery procedure.

A minimum of 2 retries, either from a retargetting or a name discovery failure. This will give the session service a chance to re-establish a NetBIOS Listen or, more importantly, allow the NetBIOS scope, local name service or the NBNS, to re-learn the correct IP address of a NetBIOS name.

The stack method operates similarly to the straight method. However, instead of retrying at the initial node returned by the name discovery procedure, it restarts with the IP address of the last node which sent a SESSION RETARGET RESPONSE prior to the retargetting failure. To limit the stack method, any one host can only be tried a maximum of 2 times.

B-5. NBDD SERVICE

If the NBDD does not forward datagrams then don't provide Group and Broadcast NetBIOS datagram services to the NetBIOS user. Therefore, ignore the implementation of the query request and, when get a negative response, acquiring the membership list of IP addresses and sending the datagram as a unicast to each member.

B-6. APPLICATION CONSIDERATIONS

B-6.1 USE OF NetBIOS DATAGRAMS

Certain existing NetBIOS applications use NetBIOS datagrams as a foundation for their own connection-oriented protocols. This can cause excessive NetBIOS name query activity and place a substantial burden on the network, server nodes, and other end- nodes. It is recommended that this practice be avoided in new applications.

Network Working Group
Request for Comments: 1002 March, 1987

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS

ABSTRACT

This RFC defines a proposed standard protocol to support NetBIOS services in a TCP/IP environment. Both local network and internet operation are supported. Various node types are defined to accommodate local and internet topologies and to allow operation with or without the use of IP broadcast.

This RFC gives the detailed specifications of the NetBIOS-over-TCP packets, protocols, and defined constants and variables. A more general overview is found in a companion RFC, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods".

TABLE OF CONTENTS

PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS

1. STATUS OF THIS MEMO

This RFC specifies a proposed standard for the DARPA Internet community. Since this topic is new to the Internet community, discussions and suggestions are specifically requested.

Please send written comments to:

Karl Auerbach
Epilogue Technology Corporation
P.O. Box 5432
Redwood City, CA 94063

Please send online comments to:

Avnish Aggarwal
Internet: mtxinu!excelan!avnish@ucbvax.berkeley.edu
Usenet: ucbvax!mtxinu!excelan!avnish

Distribution of this memorandum is unlimited.

2. ACKNOWLEDGEMENTS

This RFC has been developed under the auspices of the Internet Activities Board.

The following individuals have contributed to the development of this RFC:

Avnish Aggarwal	Arvind Agrawal	Lorenzo Aguilar
Geoffrey Arnold	Karl Auerbach	K. Ramesh Babu
Keith Ball	Amatzia Ben-Artzi	Vint Cerf
Richard Cherry	David Crocker	Steve Deering
Greg Ennis	Steve Holmgren	Jay Israel
David Kaufman	Lee LaBarre	James Lau
Dan Lynch	Gaylord Miyata	David Stevens
Steve Thomas	Ishan Wu	

The system proposed by this RFC does not reflect any existing Netbios-over-TCP implementation. However, the design incorporates considerable knowledge obtained from prior implementations. Special thanks goes to the following organizations which have provided this invaluable information:

CMC/Syros Excelan Sytek Ungermann-Bass

3. INTRODUCTION

This RFC contains the detailed packet formats and protocol specifications for NetBIOS-over-TCP. This RFC is a companion to RFC 1001, "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods" [1].

4. PACKET DESCRIPTIONS

Bit and byte ordering are defined by the most recent version of "Assigned Numbers" [2].

4.1. NAME FORMAT

The NetBIOS name representation in all NetBIOS packets (for NAME, SESSION, and DATAGRAM services) is defined in the Domain Name Service RFC 883[3] as "compressed" name messages. This format is called "second-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

For ease of description, the first two paragraphs from page 31, the section titled "Domain name representation and compression", of RFC 883 are replicated here:

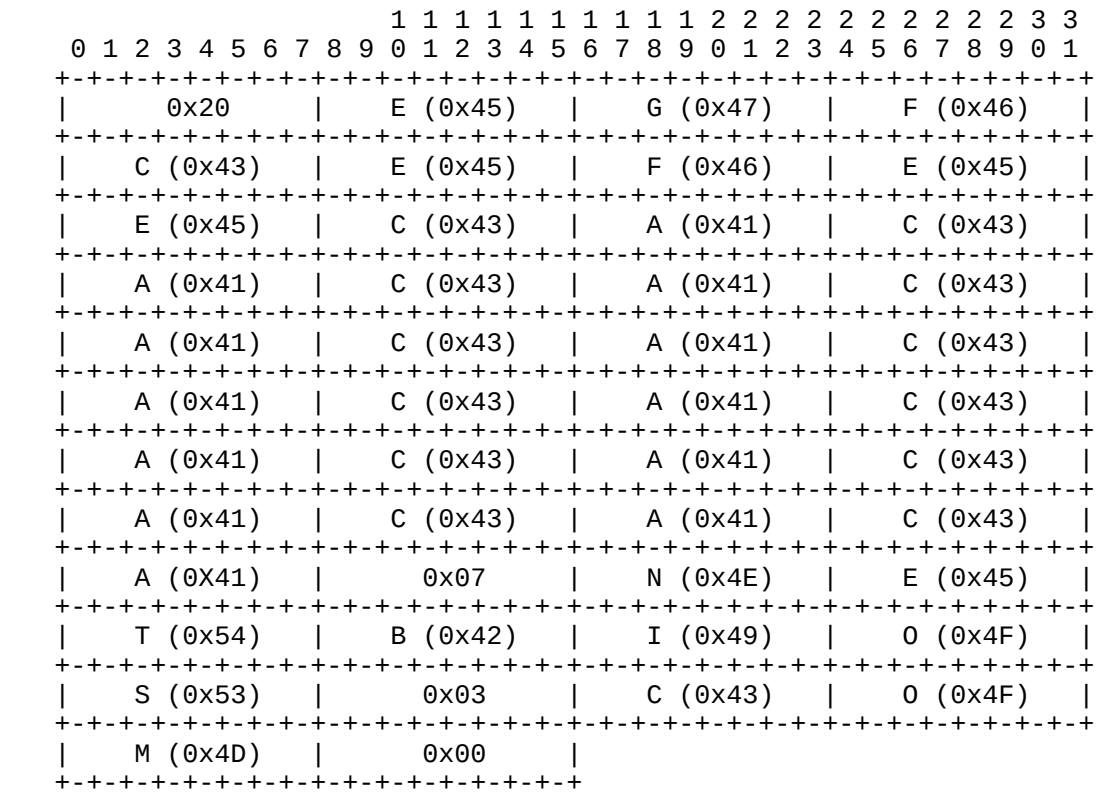
> Domain names messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a compressed domain name is terminated by a length byte of zero. The high order two bits of the length field must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.
>
> To simplify implementations, the total length of label octets and label length octets that make up a domain name is restricted to 255 octets or less.

The following is the uncompressed representation of the NetBIOS name "FRED ", which is the 4 ASCII characters, F, R, E, D, followed by 12 space characters (0x20). This name has the SCOPE_ID: "NETBIOS.COM"

```
EGFCEFEECACACACACACACACACACACACA.NETBIOS.COM
```

This uncompressed representation of names is called "first-level encoding" in the section entitled "Representation of NetBIOS Names" in the Concepts and Methods document.

The following is a pictographic representation of the compressed representation of the previous uncompressed Domain Name representation.



Each section of a domain name is called a label [7 (page 31)]. A label can be a maximum of 63 bytes. The first byte of a label in compressed representation is the number of bytes in the label. For the above example, the first 0x20 is the number of bytes in the left-most label, EGFCEFEECACACACACACACACACACACA, of the domain name. The bytes following the label length count are the characters of the label. The following labels are in sequence after the first label, which is the encoded NetBIOS name, until a zero (0x00) length count. The zero length count represents the root label, which is always null.

A label length count is actually a 6-bit field in the label length field. The most significant 2 bits of the field, bits 7 and 6, are flags allowing an escape from the above compressed representation. If bits 7 and 6 are both set (11), the following 14 bits are an offset pointer into the full message to the actual label string from another domain name that belongs in this name. This label pointer allows for a further compression of a domain name in a packet.

NetBIOS implementations can only use label string pointers in Name Service packets. They cannot be used in Session or Datagram Service packets.

The other two possible values for bits 7 and 6 (01 and 10) of a label length field are reserved for future use by RFC 883[2 (page 32)].

Note that the first octet of a compressed name must contain one of the following bit patterns. (An "x" indicates a bit whose value may be either 0 or 1.):

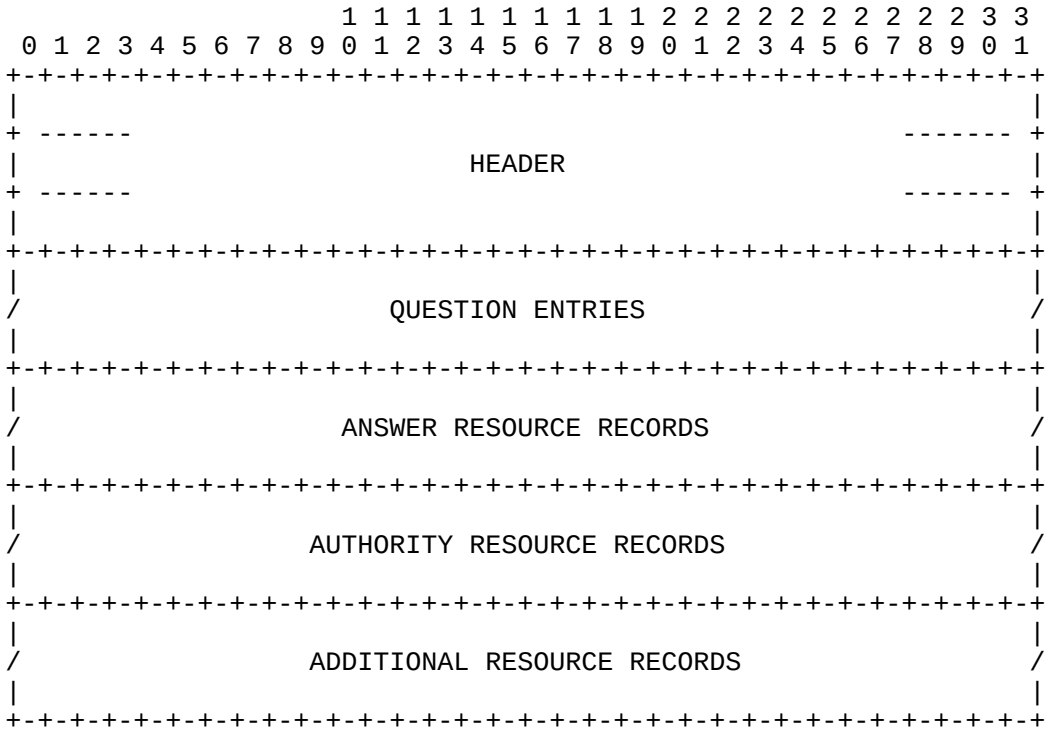
```
00100000 -  Netbios name, length must be 32 (decimal)
11xxxxxx -  Label string pointer
10xxxxxx -  Reserved
01xxxxxx -  Reserved
```

4.2. NAME SERVICE PACKETS

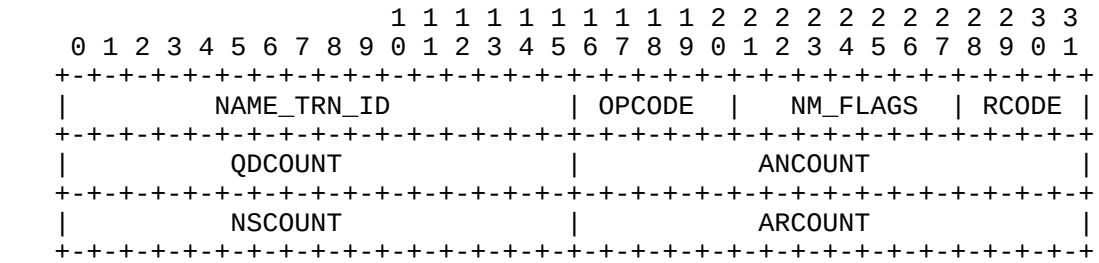
4.2.1. GENERAL FORMAT OF NAME SERVICE PACKETS

The NetBIOS Name Service packets follow the packet structure defined in the Domain Name Service (DNS) RFC 883 [7 (pg 26-31)]. The structures are compatible with the existing DNS packet formats, however, additional types and codes have been added to work with NetBIOS.

If Name Service packets are sent over a TCP connection they are preceded by a 16 bit unsigned integer representing the length of the Name Service packet.

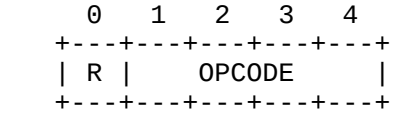


4.2.1.1. HEADER



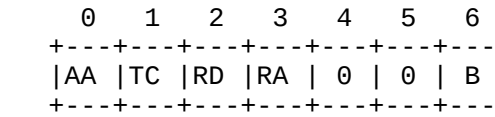
Field	Description
NAME_TRN_ID	Transaction ID for Name Service Transaction. Requestor places a unique value for each active transaction. Responder puts NAME_TRN_ID value from request packet in response packet.
OPCODE	Packet type code, see table below.
NM_FLAGS	Flags for operation, see table below.
RCODE	Result codes of request. Table of RCODE values for each response packet below.
QDCOUNT	Unsigned 16 bit integer specifying the number of entries in the question section of a Name Service packet. Always zero (0) for responses. Must be non-zero for all NetBIOS Name requests.
ANCOUNT	Unsigned 16 bit integer specifying the number of resource records in the answer section of a Name Service packet.
NSCOUNT	Unsigned 16 bit integer specifying the number of resource records in the authority section of a Name Service packet.
ARCOUNT	Unsigned 16 bit integer specifying the number of resource records in the additional records section of a Name Service packet.

The OPCODE field is defined as:



Symbol	Bit(s)	Description
OPCODE	1-4	Operation specifier: 0 = query 5 = registration 6 = release 7 = WACK 8 = refresh
R	0	RESPONSE flag: if bit == 0 then request packet if bit == 1 then response packet.

The NM_FLAGS field is defined as:

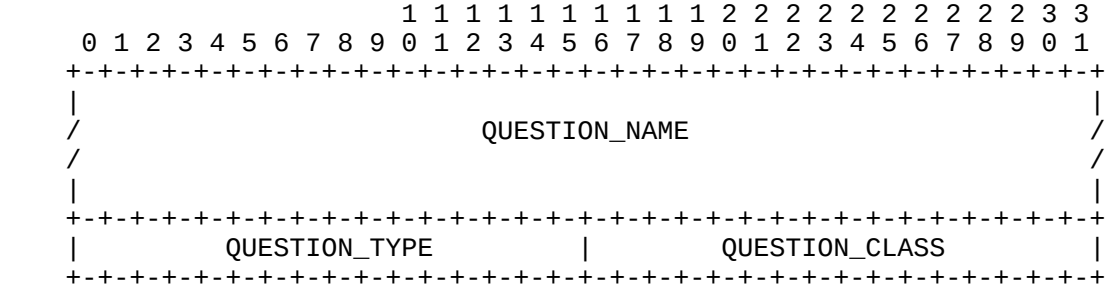


Symbol	Bit(s)	Description
B	6	Broadcast Flag. = 1: packet was broadcast or multicast = 0: unicast
RA	3	Recursion Available Flag. Only valid in responses from a NetBIOS Name Server -- must be zero in all other responses. If one (1) then the NBNS supports recursive query, registration, and release. If zero (0) then the end-node must iterate for query and challenge for registration.
RD	2	Recursion Desired Flag. May only be set on a request to a NetBIOS Name Server. The NBNS will copy its state into the response packet. If one (1) the NBNS will iterate on the query, registration, or release.
TC	1	Truncation Flag.

Set if this message was truncated because the datagram carrying it would be greater than 576 bytes in length. Use TCP to get the information from the NetBIOS Name Server.

AA	0	Authoritative Answer flag. Must be zero (0) if R flag of OPCODE is zero (0). If R flag is one (1) then if AA is one (1) then the node responding is an authority for the domain name. End nodes responding to queries always set this bit in responses.

4.2.1.2. QUESTION SECTION



Field	Description
QUESTION_NAME	The compressed name representation of the NetBIOS name for the request.
QUESTION_TYPE	The type of request. The values for this field are specified for each request.
QUESTION_CLASS	The class of the request. The values for this field are specified for each request.

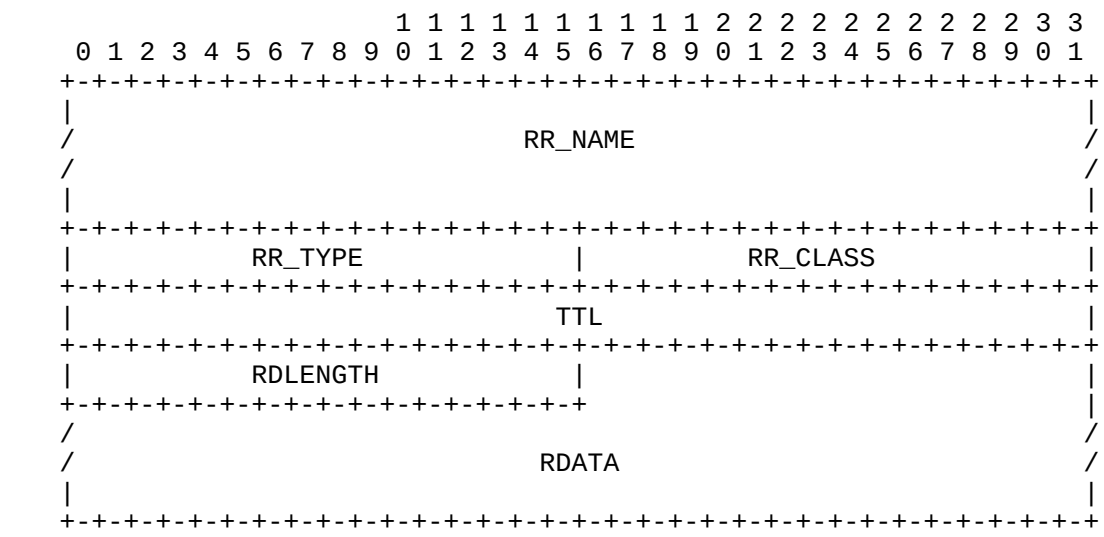
QUESTION_TYPE is defined as:

Symbol	Value	Description:
NB	0x0020	NetBIOS general Name Service Resource Record
NBSTAT	0x0021	NetBIOS NODE STATUS Resource Record (See NODE STATUS REQUEST)

QUESTION_CLASS is defined as:

Symbol	Value	Description:
IN	0x0001	Internet class

4.2.1.3. RESOURCE RECORD



Field	Description
RR_NAME	The compressed name representation of the NetBIOS name corresponding to this resource record.
RR_TYPE	Resource record type code
RR_CLASS	Resource record class code
TTL	The Time To Live of a the resource record’s name.
RDLENGTH	Unsigned 16 bit integer that specifies the number of bytes in the RDATA field.
RDATA	RR_CLASS and RR_TYPE dependent field. Contains the resource information for the NetBIOS name.

RESOURCE RECORD RR_TYPE field definitions:

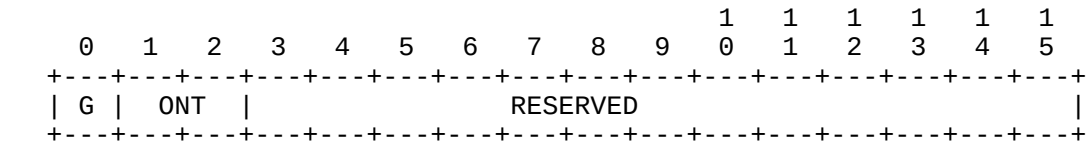
Symbol	Value	Description:
A	0x0001	IP address Resource Record (See REDIRECT NAME QUERY RESPONSE)
NS	0x0002	Name Server Resource Record (See REDIRECT

NAME QUERY RESPONSE)
NULL 0x000A NULL Resource Record (See WAIT FOR ACKNOWLEDGEMENT RESPONSE)
NB 0x0020 NetBIOS general Name Service Resource Record (See NB_FLAGS and NB_ADDRESS, below)
NBSTAT 0x0021 NetBIOS NODE STATUS Resource Record (See NODE STATUS RESPONSE)

RESOURCE RECORD RR_CLASS field definitions:

Symbol	Value	Description:
IN	0x0001	Internet class

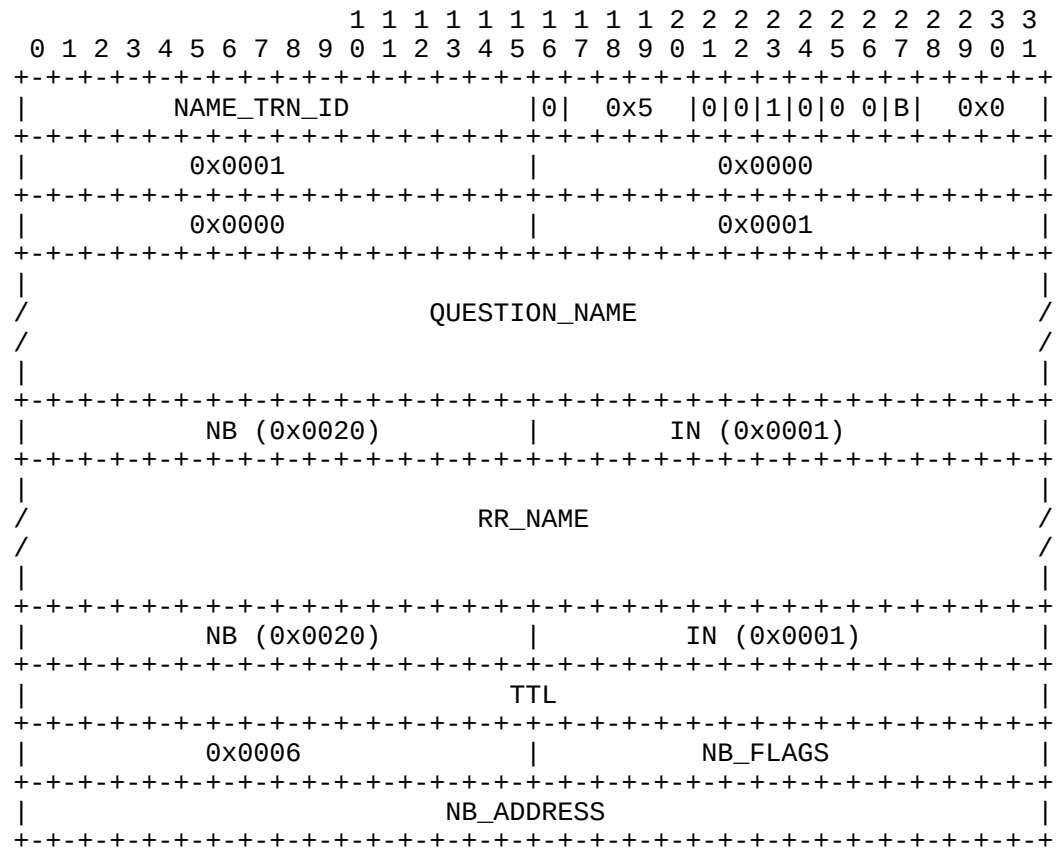
NB_FLAGS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB":



Symbol	Bit(s)	Description:
RESERVED	3-15	Reserved for future use. Must be zero (0).
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use For registration requests this is the claimant's type. For responses this is the actual owner's type.
G	0	Group Name Flag. If one (1) then the RR_NAME is a GROUP NetBIOS name. If zero (0) then the RR_NAME is a UNIQUE NetBIOS name.

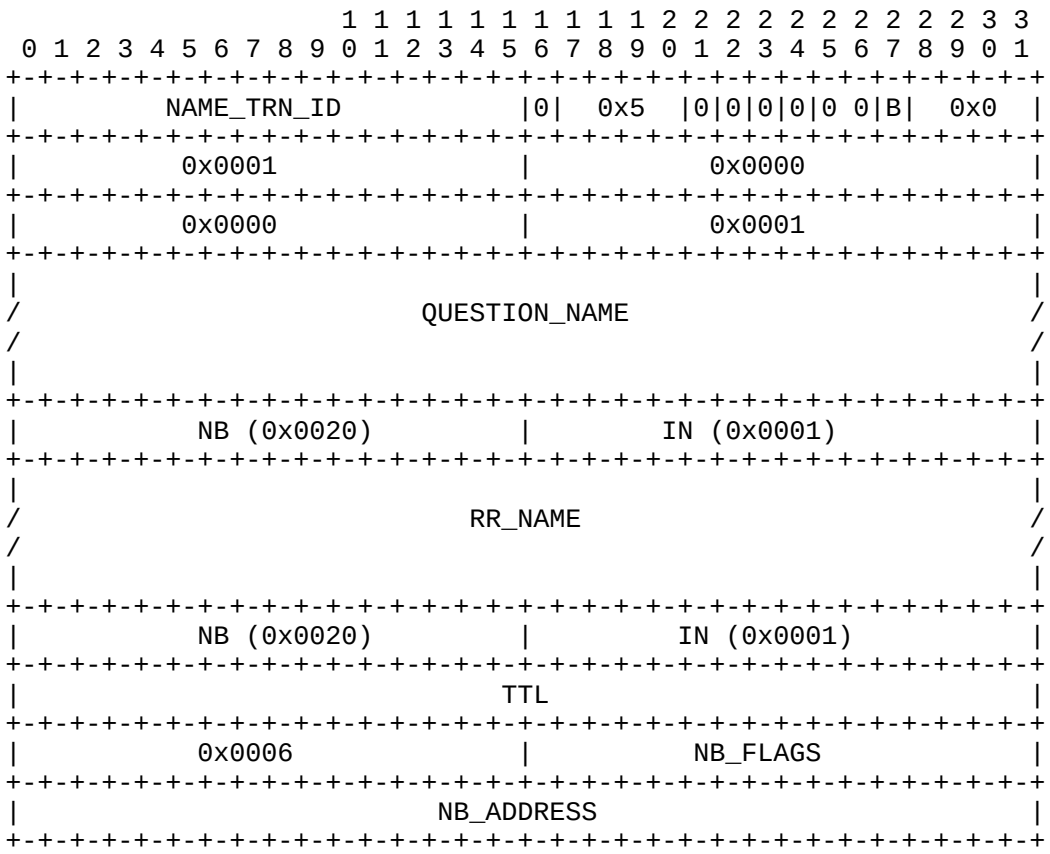
The NB_ADDRESS field of the RESOURCE RECORD RDATA field for RR_TYPE of "NB" is the IP address of the name's owner.

4.2.2. NAME REGISTRATION REQUEST

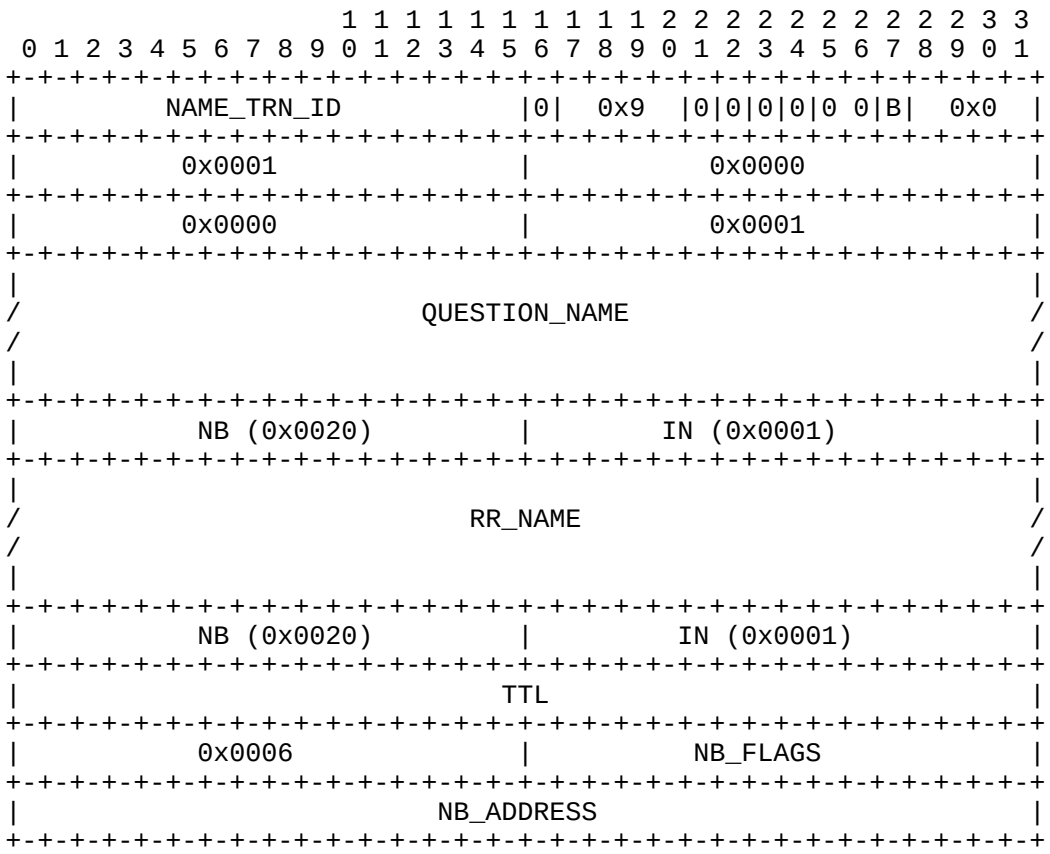


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use pointers to the QUESTION_NAME name's labels to guarantee the length of the datagram is less than the maximum 576 bytes. See section above on name formats and also page 31 and 32 of RFC 883, Domain Names - Implementation and Specification, for a complete description of compressed name label pointers.

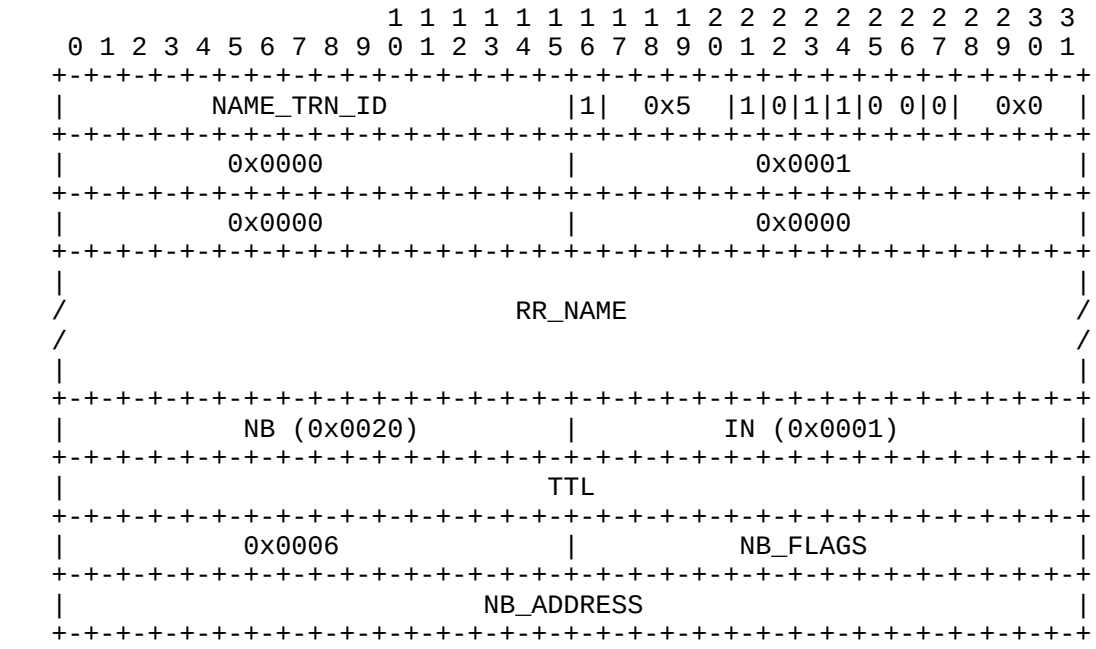
4.2.3. NAME OVERWRITE REQUEST & DEMAND



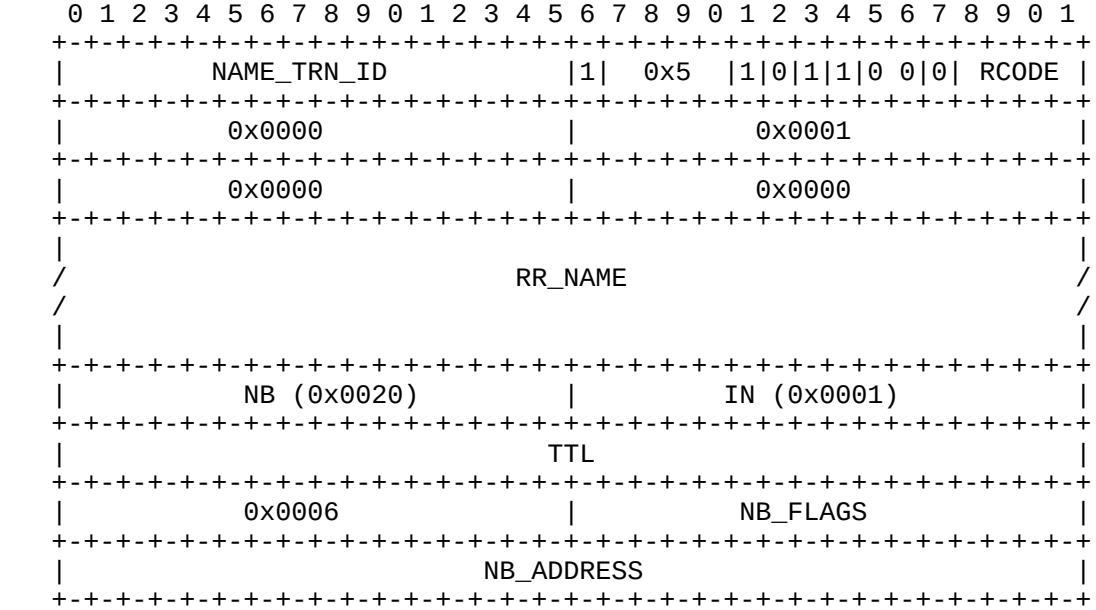
4.2.4. NAME REFRESH REQUEST



4.2.5. POSITIVE NAME REGISTRATION RESPONSE



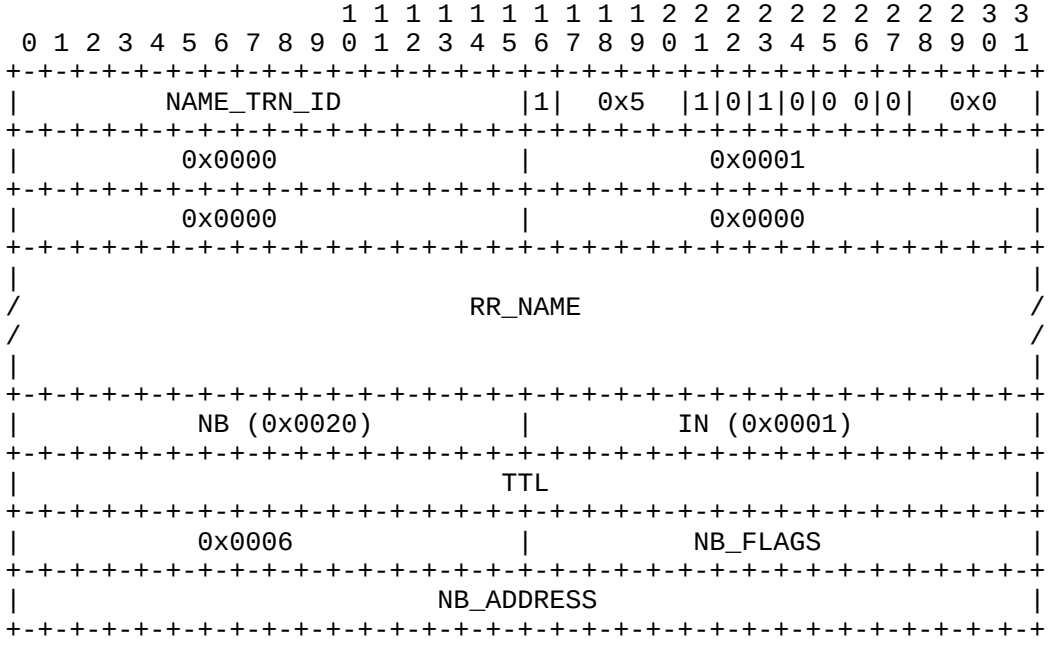
4.2.6. NEGATIVE NAME REGISTRATION RESPONSE



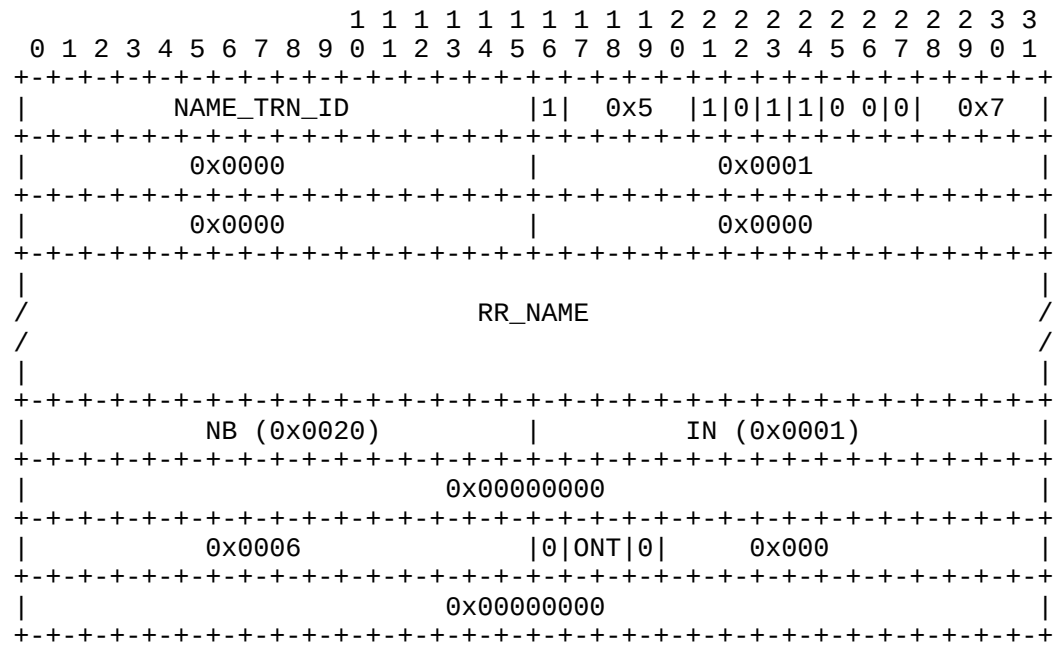
RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node.
CFT_ERR	0x7	Name in conflict error. A UNIQUE name is owned by more than one node.

4.2.7. END-NODE CHALLENGE REGISTRATION RESPONSE

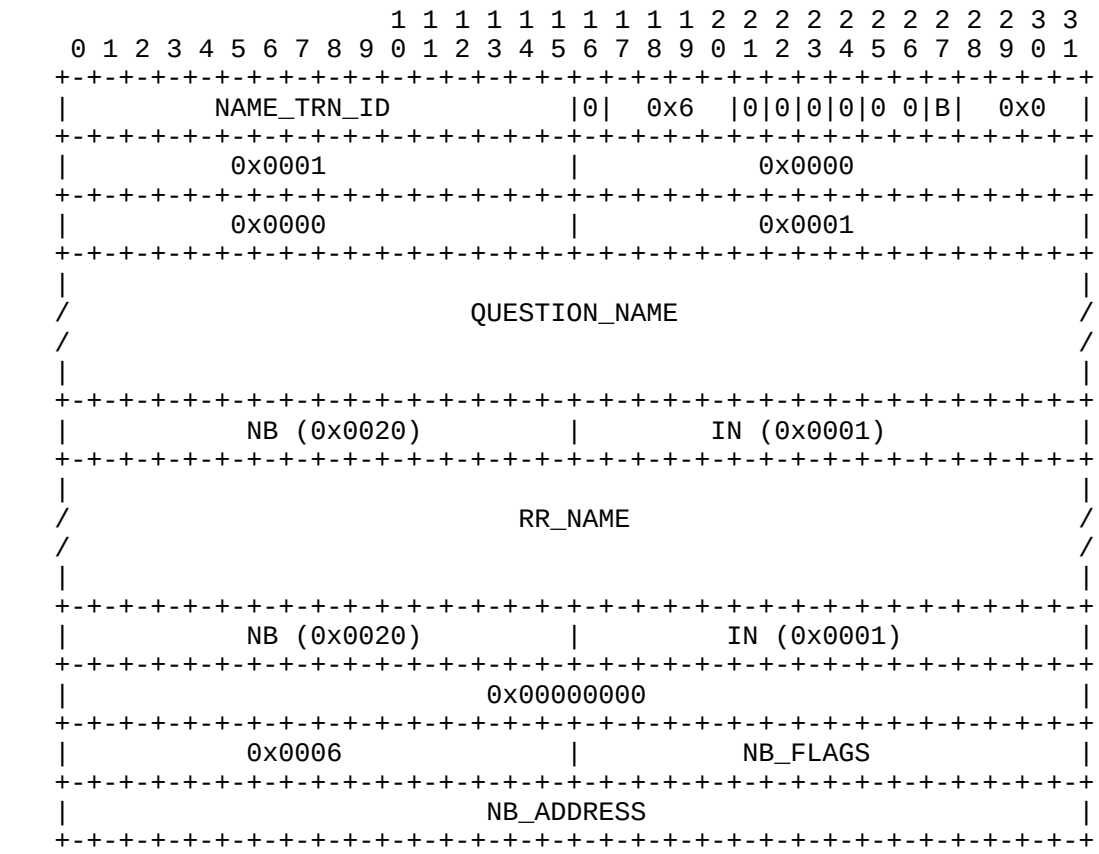


4.2.8. NAME CONFLICT DEMAND



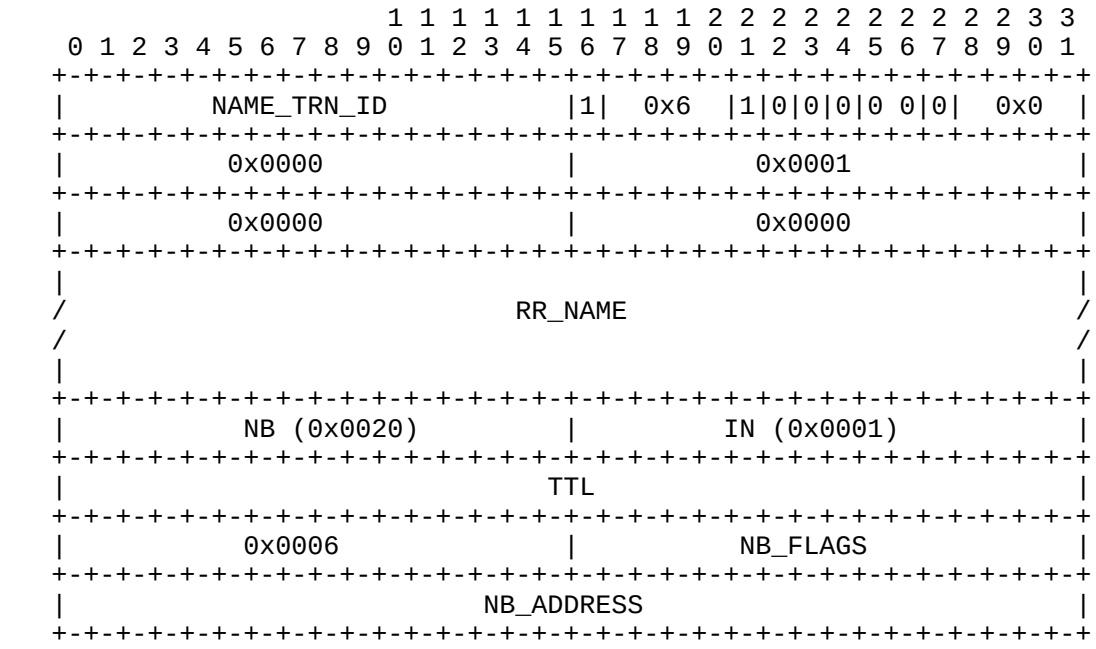
This packet is identical to a NEGATIVE NAME REGISTRATION RESPONSE with RCODE = CFT_ERR.

4.2.9. NAME RELEASE REQUEST & DEMAND

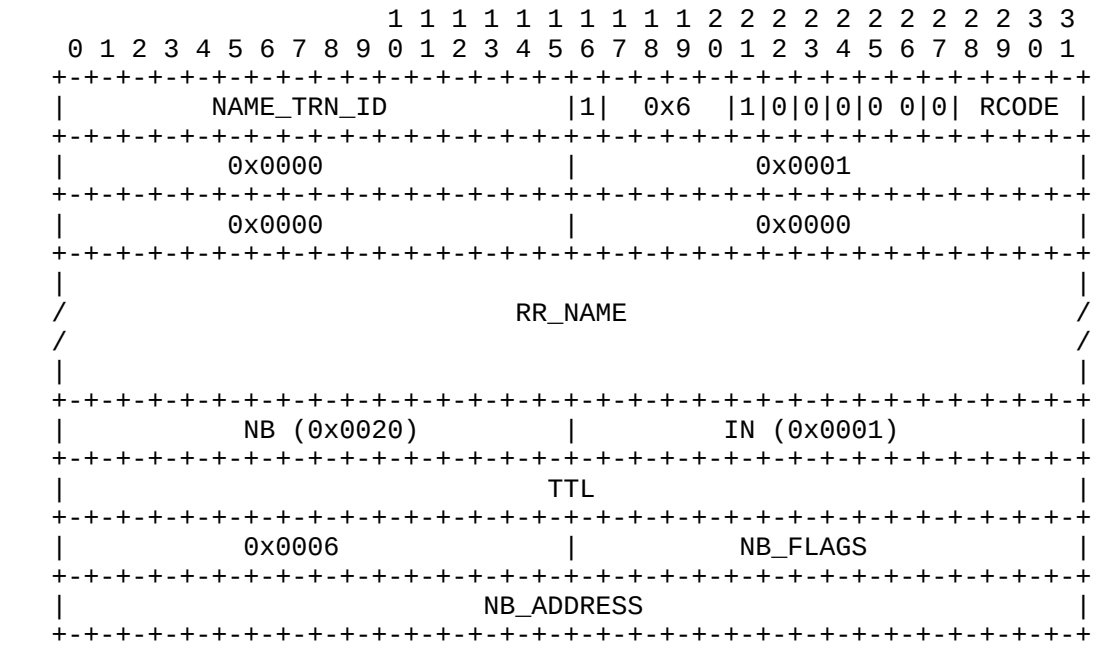


Since the RR_NAME is the same name as the QUESTION_NAME, the RR_NAME representation must use label string pointers to the QUESTION_NAME labels to guarantee the length of the datagram is less than the maximum 576 bytes. This is the same condition as with the NAME REGISTRATION REQUEST.

4.2.10. POSITIVE NAME RELEASE RESPONSE



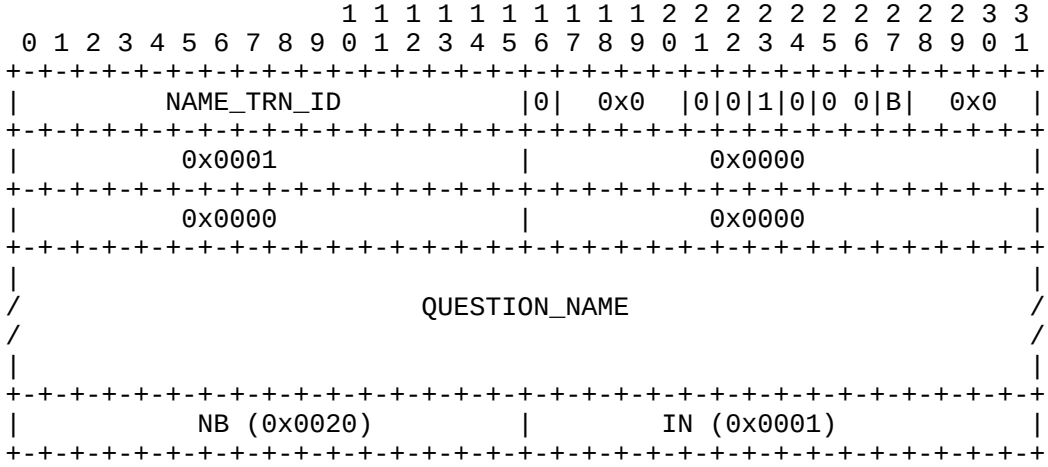
4.2.11. NEGATIVE NAME RELEASE RESPONSE



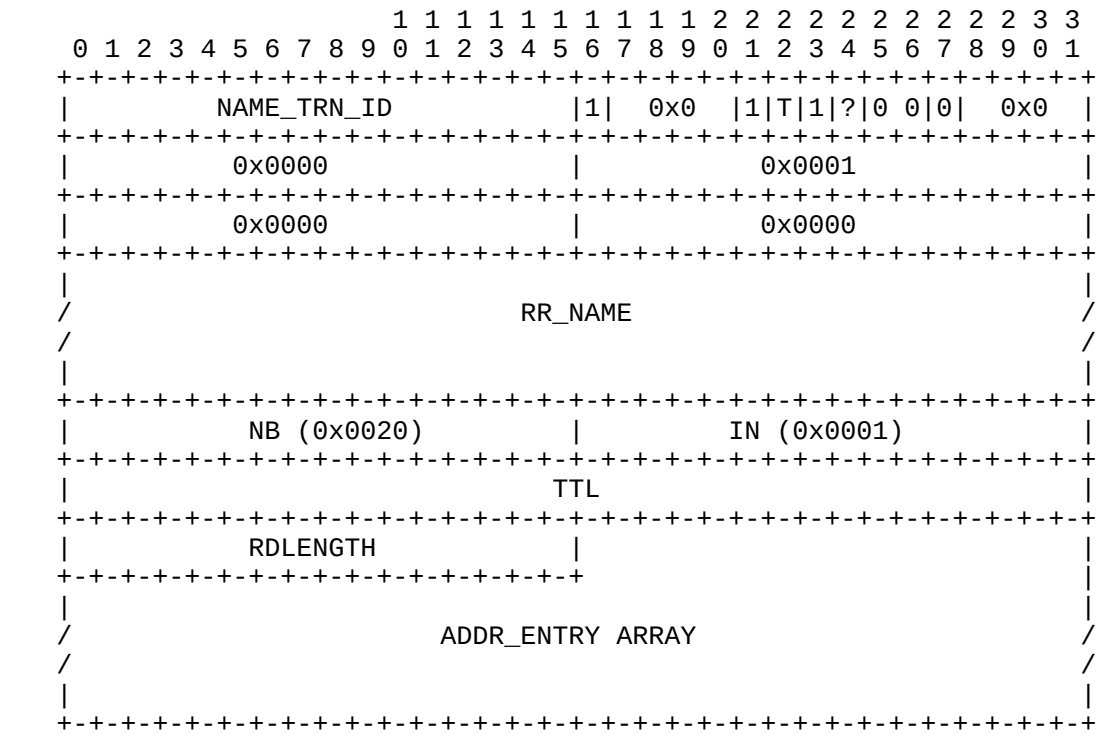
RCODE field values:

Symbol	Value	Description:
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
RFS_ERR	0x5	Refused error. For policy reasons server will not release this name from this host.
ACT_ERR	0x6	Active error. Name is owned by another node. Only that node may release it. A NetBIOS Name Server can optionally allow a node to release a name it does not own. This would facilitate detection of inactive names for nodes that went down silently.

4.2.12. NAME QUERY REQUEST

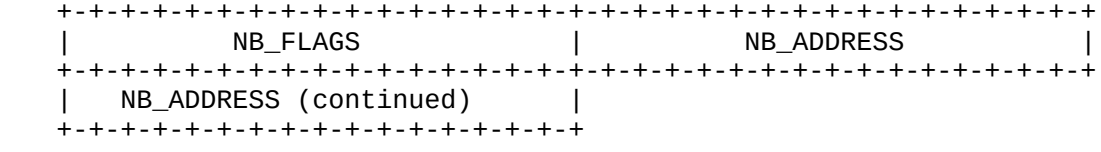


4.2.13. POSITIVE NAME QUERY RESPONSE

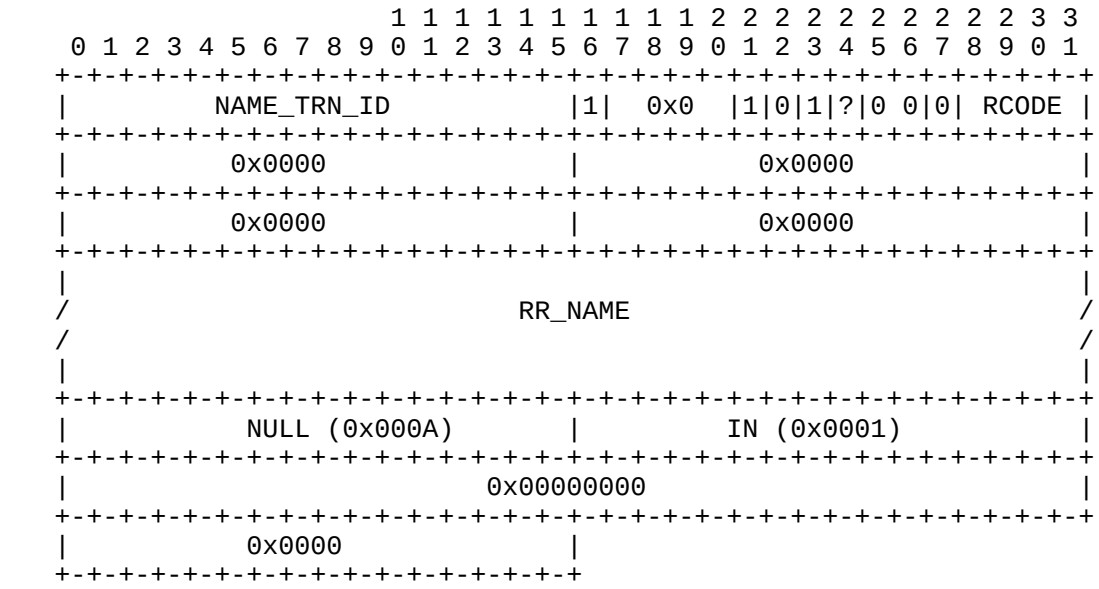


The ADDR_ENTRY ARRAY a sequence of zero or more ADDR_ENTRY records. Each ADDR_ENTRY record represents an owner of a name. For group names there may be multiple entries. However, the list may be incomplete due to packet size limitations. Bit 22, "T", will be set to indicate truncated data.

Each ADDR_ENTRY has the following format:



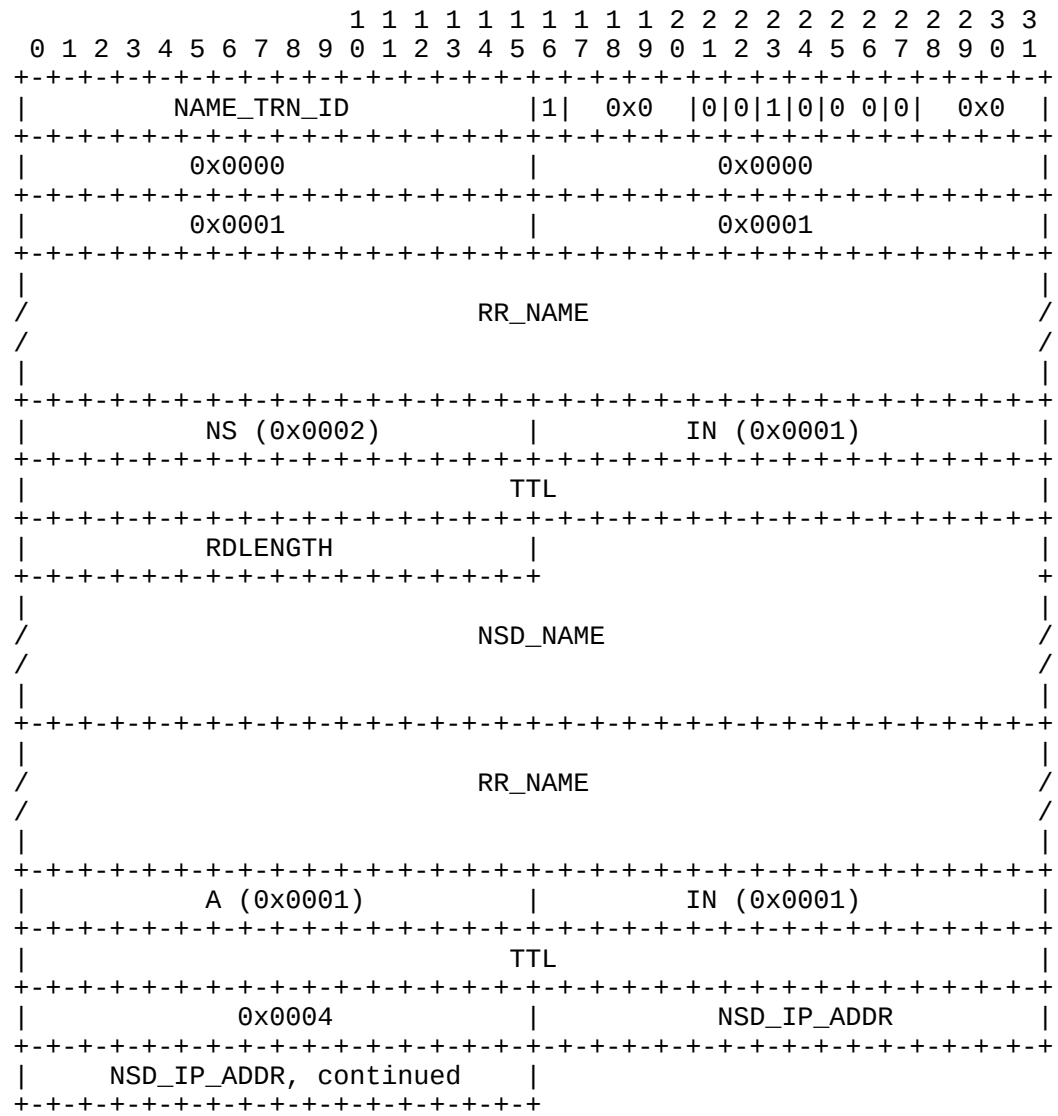
4.2.14. NEGATIVE NAME QUERY RESPONSE



RCODE field values:

Symbol	Value	Description
FMT_ERR	0x1	Format Error. Request was invalidly formatted.
SRV_ERR	0x2	Server failure. Problem with NBNS, cannot process name.
NAM_ERR	0x3	Name Error. The name requested does not exist.
IMP_ERR	0x4	Unsupported request error. Allowable only for challenging NBNS when gets an Update type registration request.
RFS_ERR	0x5	Refused error. For policy reasons server will not register this name from this host.

4.2.15. REDIRECT NAME QUERY RESPONSE



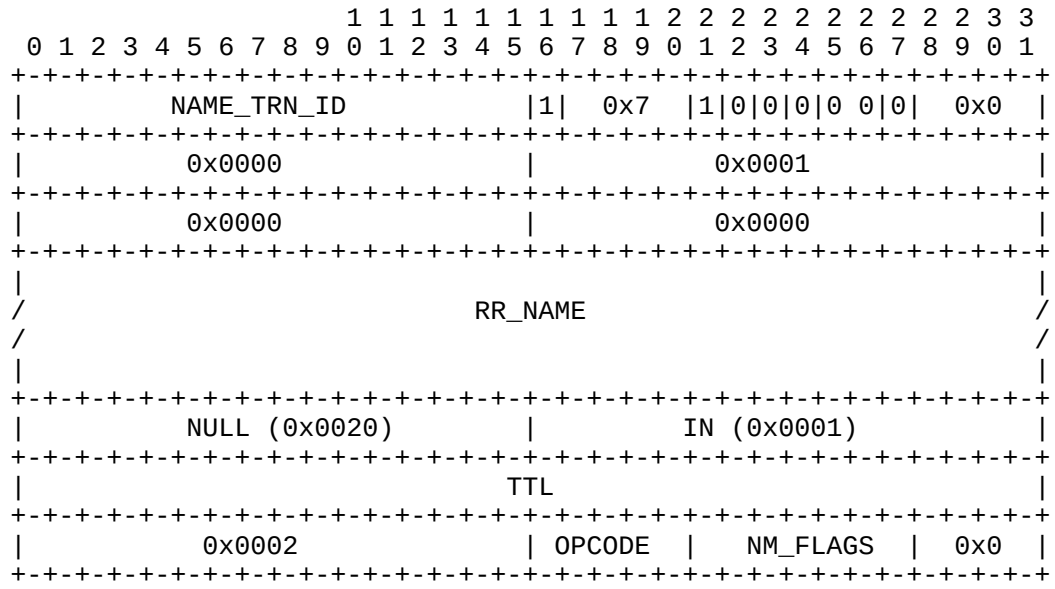
An end node responding to a NAME QUERY REQUEST always responds with the AA and RA bits set for both the NEGATIVE and POSITIVE NAME QUERY RESPONSE packets. An end node never sends a REDIRECT NAME QUERY RESPONSE packet.

When the requestor receives the REDIRECT NAME QUERY RESPONSE it must reiterate the NAME QUERY REQUEST to the NBNS specified by the NSD_IP_ADDR field of the A type RESOURCE RECORD in the ADDITIONAL section of the response packet. This is an optional packet for the NBNS.

The NSD_NAME and the RR_NAME in the ADDITIONAL section of the response packet are the same name. Space can be optimized if label string pointers are used in the RR_NAME which point to the labels in the NSD_NAME.

The RR_NAME in the AUTHORITY section is the name of the domain the NBNS called by NSD_NAME has authority over.

4.2.16. WAIT FOR ACKNOWLEDGEMENT (WACK) RESPONSE

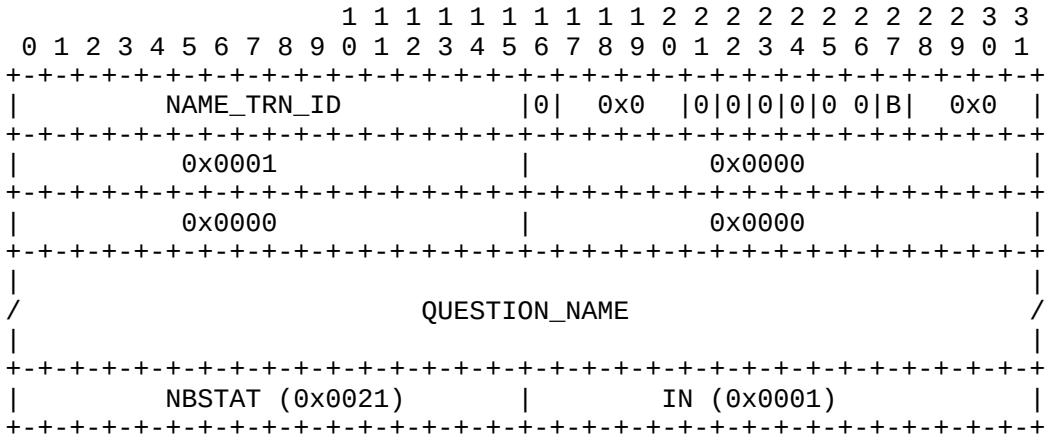


The NAME_TRN_ID of the WACK RESPONSE packet is the same NAME_TRN_ID of the request that the NBNS is telling the requestor to wait longer to complete. The RR_NAME is the name from the request, if any. If no name is available from the request then it is a null name, single byte of zero.

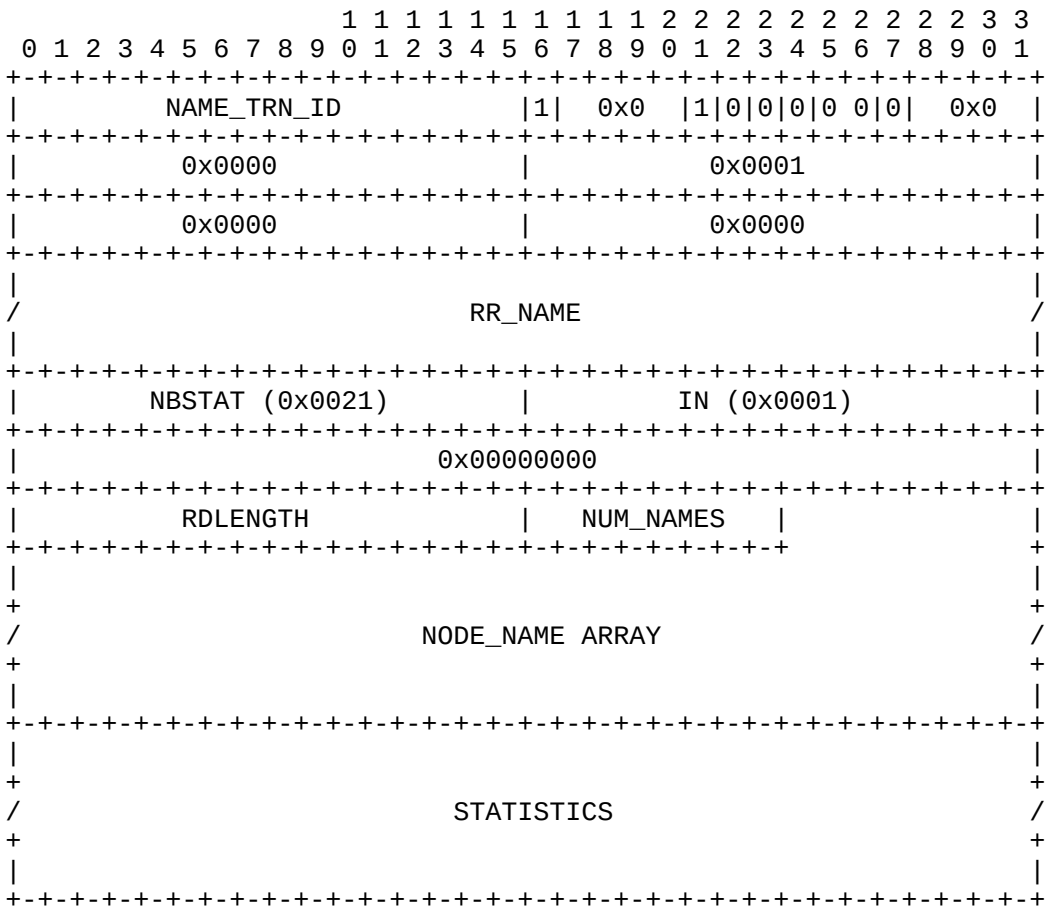
The TTL field of the ResourceRecord is the new time to wait, in seconds, for the request to complete. The RDATA field contains the OPCODE and NM_FLAGS of the request.

A TTL value of 0 means that the NBNS can not estimate the time it may take to complete a response.

4.2.17. NODE STATUS REQUEST

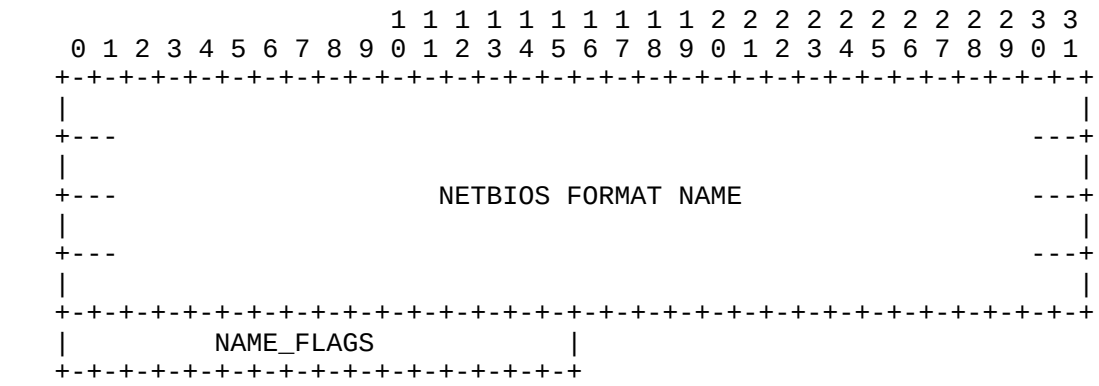


4.2.18. NODE STATUS RESPONSE

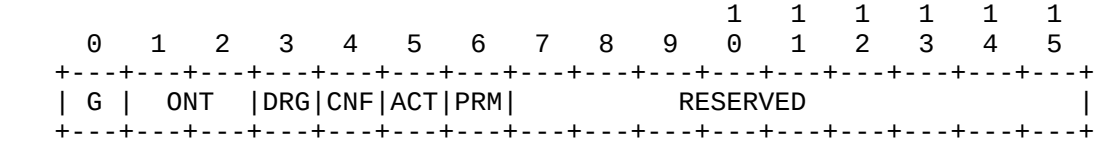


The NODE_NAME ARRAY is an array of zero or more NUM_NAMES entries of NODE_NAME records. Each NODE_NAME entry represents an active name in the same NetBIOS scope as the requesting name in the local name table of the responder. RR_NAME is the requesting name.

NODE_NAME Entry:



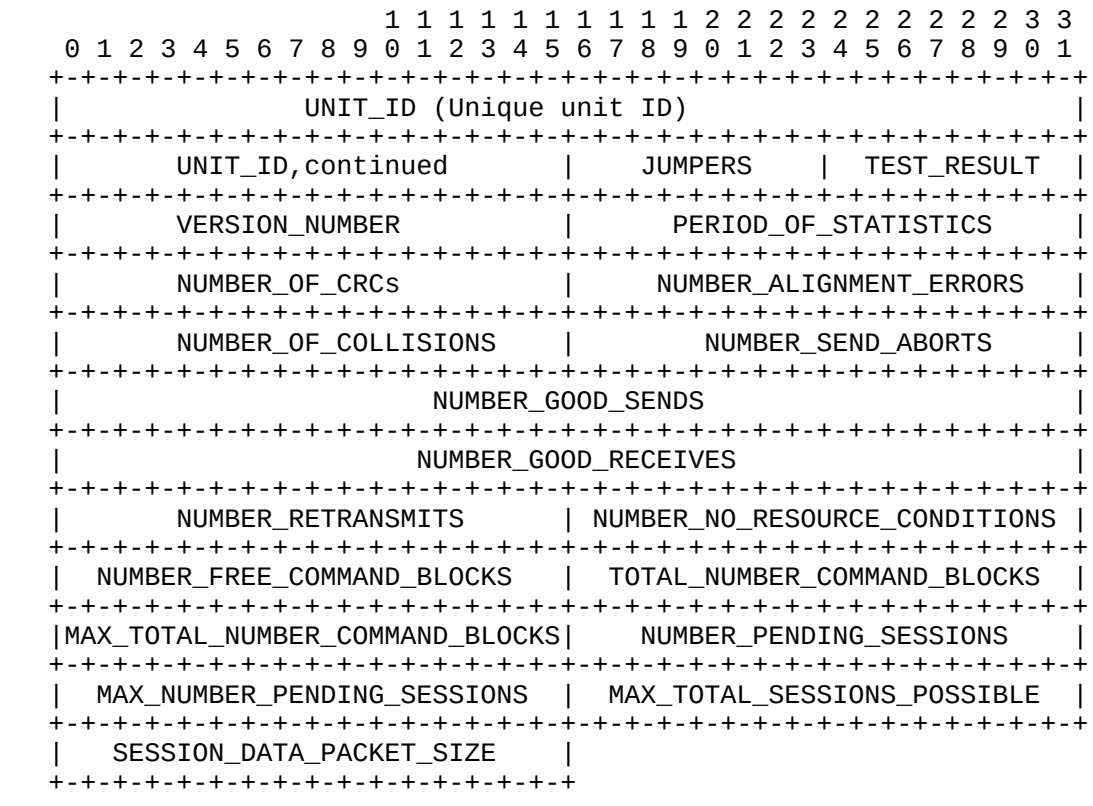
The NAME_FLAGS field:



The NAME_FLAGS field is defined as:

Symbol	Bit(s)	Description:
RESERVED	7-15	Reserved for future use. Must be zero (0).
PRM	6	Permanent Name Flag. If one (1) then entry is for the permanent node name. Flag is zero (0) for all other names.
ACT	5	Active Name Flag. All entries have this flag set to one (1).
CNF	4	Conflict Flag. If one (1) then name on this node is in conflict.
DRG	3	Deregister Flag. If one (1) then this name is in the process of being deleted.
ONT	1,2	Owner Node Type: 00 = B node 01 = P node 10 = M node 11 = Reserved for future use
G	0	Group Name Flag. If one (1) then the name is a GROUP NetBIOS name. If zero (0) then it is a UNIQUE NetBIOS name.

STATISTICS Field of the NODE STATUS RESPONSE:

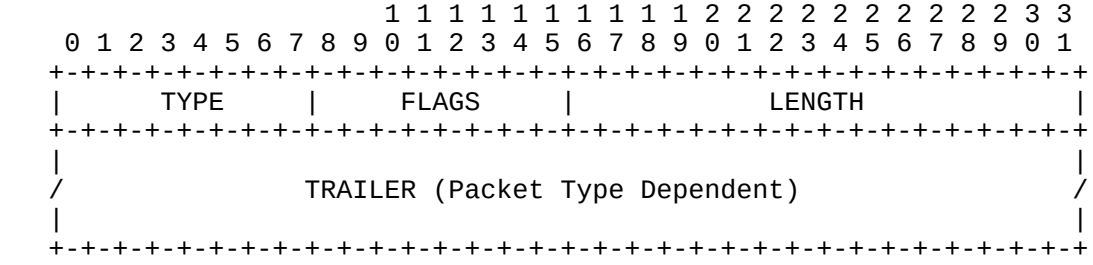


4.3. SESSION SERVICE PACKETS

4.3.1. GENERAL FORMAT OF SESSION PACKETS

All session service messages are sent over a TCP connection.

All session packets are of the following general structure:



The TYPE, FLAGS, and LENGTH fields are present in every session packet.

The LENGTH field is the number of bytes following the LENGTH field. In other words, LENGTH is the combined size of the TRAILER field(s). For example, the POSITIVE SESSION RESPONSE packet always has a LENGTH field value of zero (0000) while the RETARGET SESSION RESPONSE always has a LENGTH field value of six (0006).

One of the bits of the FLAGS field acts as an additional, high-order bit for the LENGTH field. Thus the cumulative size of the trailer field(s) may range from 0 to 128K bytes.

Session Packet Types (in hexidecimal):

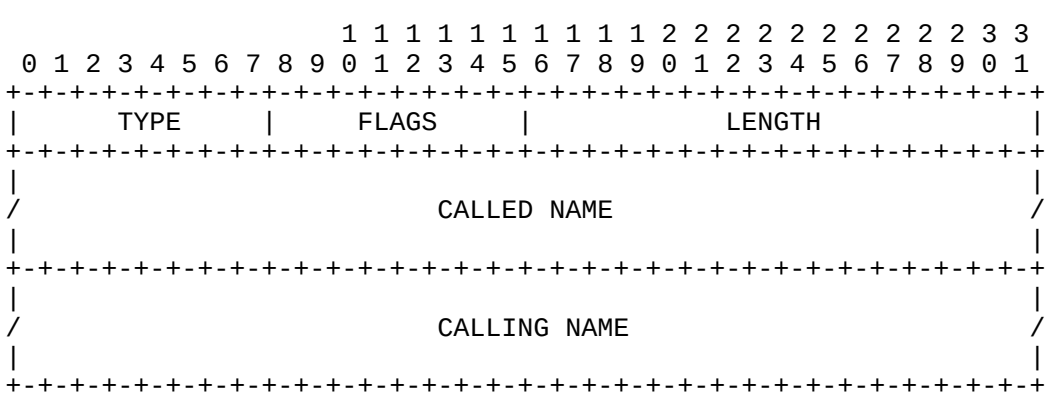
```
        00 -  SESSION MESSAGE
        81 -  SESSION REQUEST
        82 -  POSITIVE SESSION RESPONSE
        83 -  NEGATIVE SESSION RESPONSE
        84 -  RETARGET SESSION RESPONSE
        85 -  SESSION KEEP ALIVE
```

Bit definitions of the FLAGS field:

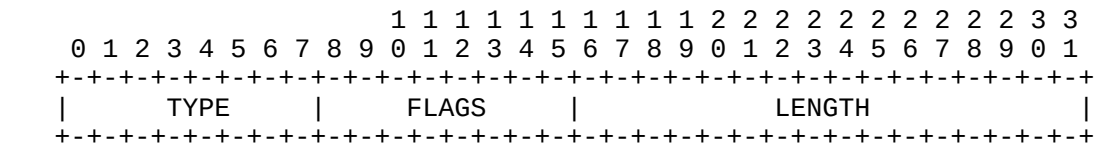
```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | E |
+---+---+---+---+---+---+---+---+
```

Symbol	Bit(s)	Description
E	7	Length extension, used as an additional, high-order bit on the LENGTH field.
RESERVED	0-6	Reserved, must be zero (0)

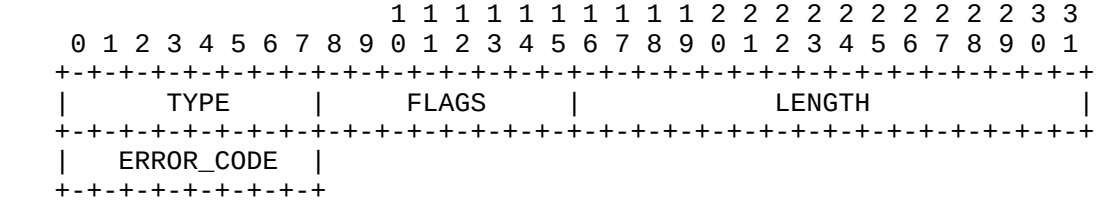
4.3.2. SESSION REQUEST PACKET



4.3.3. POSITIVE SESSION RESPONSE PACKET



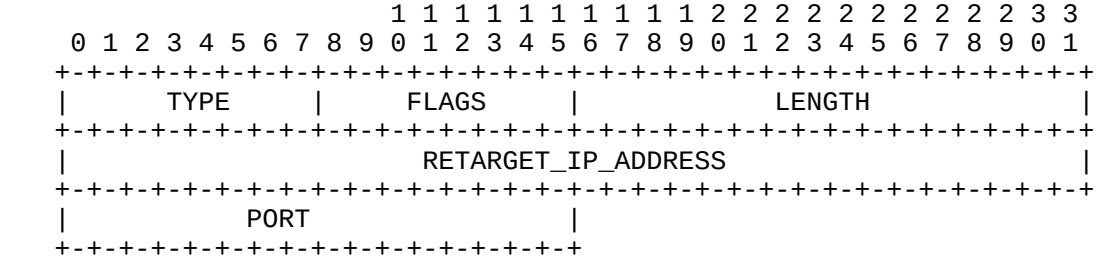
4.3.4. NEGATIVE SESSION RESPONSE PACKET



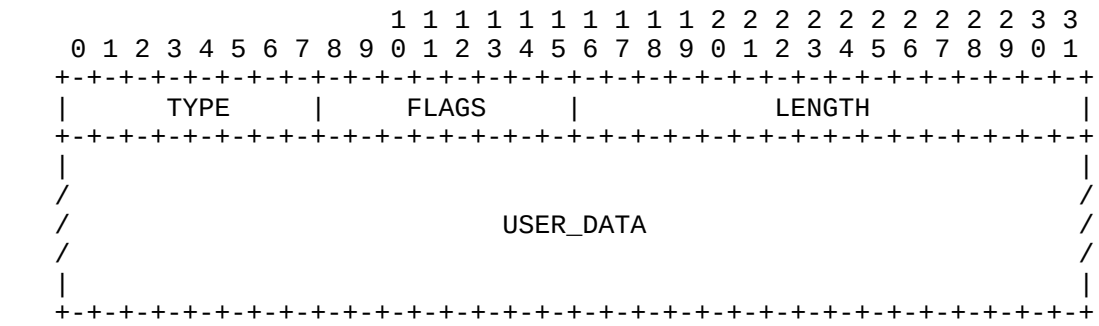
NEGATIVE SESSION RESPONSE packet error code values (in hexidecimal):

- 80 - Not listening on called name
- 81 - Not listening for calling name
- 82 - Called name not present
- 83 - Called name present, but insufficient resources
- 8F - Unspecified error

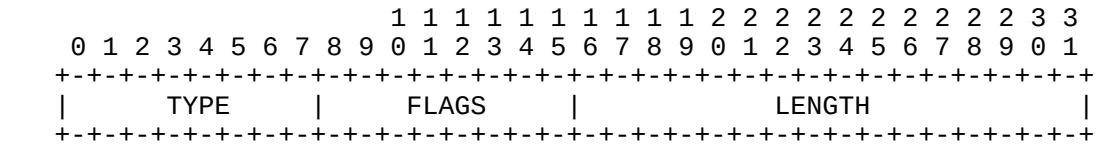
4.3.5. SESSION RETARGET RESPONSE PACKET



4.3.6. SESSION MESSAGE PACKET

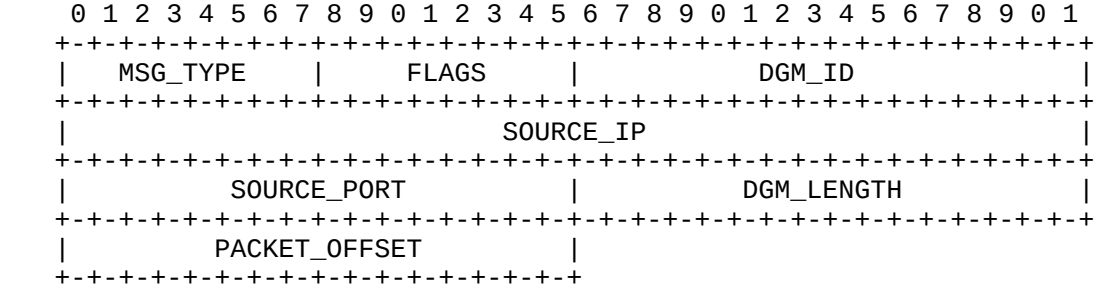


4.3.7. SESSION KEEP ALIVE PACKET



4.4. DATAGRAM SERVICE PACKETS

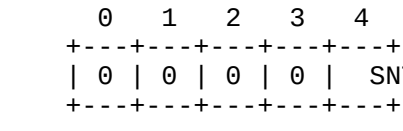
4.4.1. NetBIOS DATAGRAM HEADER



MSG_TYPE values (in hexidecimal):

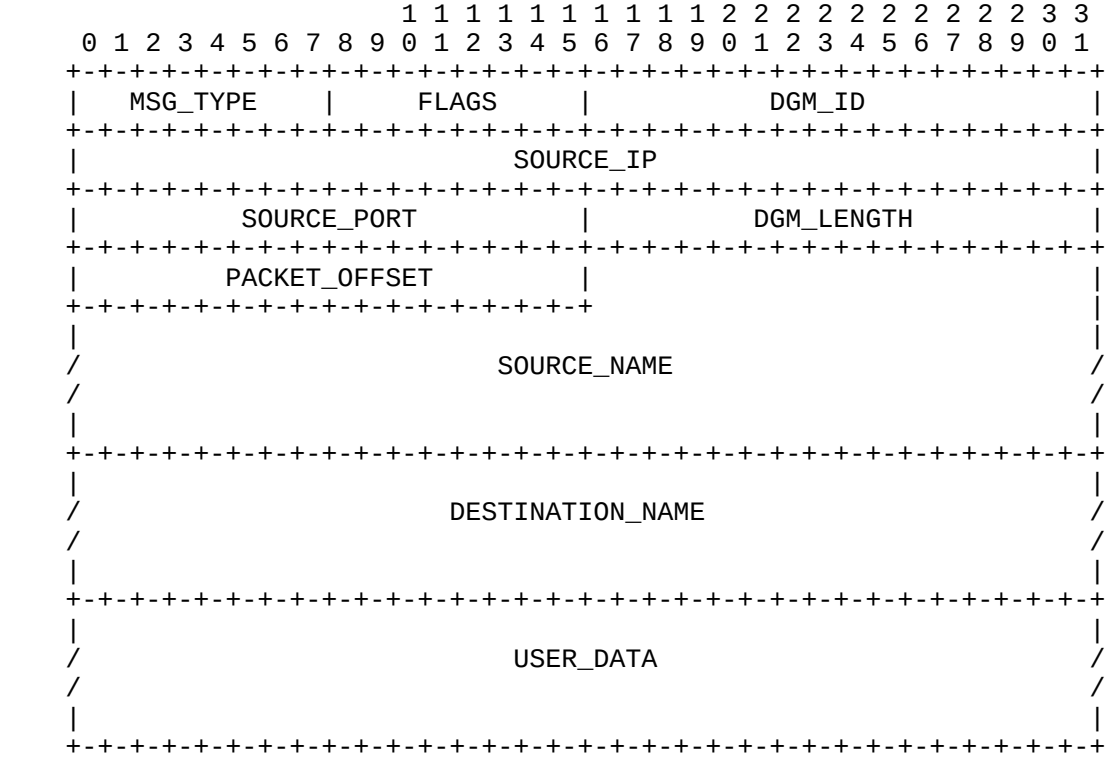
10 - DIRECT_UNIQUE DATAGRAM
11 - DIRECT_GROUP DATAGRAM
12 - BROADCAST DATAGRAM
13 - DATAGRAM ERROR
14 - DATAGRAM QUERY REQUEST
15 - DATAGRAM POSITIVE QUERY RESPONSE
16 - DATAGRAM NEGATIVE QUERY RESPONSE

Bit definitions of the FLAGS field:

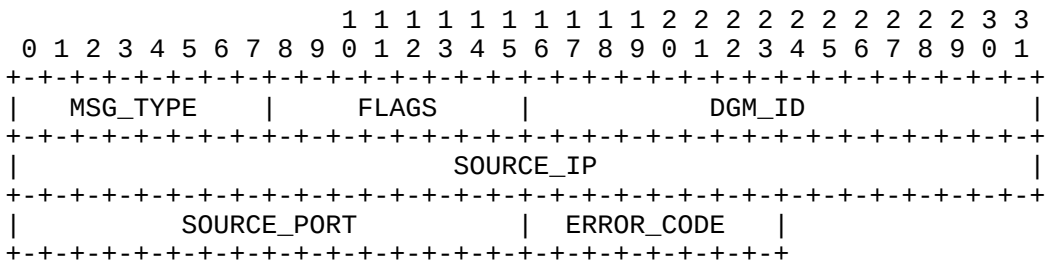


Symbol	Bit(s)	Description
M	7	MORE flag, If set then more NetBIOS datagram fragments follow.
F	6	FIRST packet flag, If set then this is first (and possibly only) fragment of NetBIOS datagram
SNT	4,5	Source End-Node type: 00 = B node 01 = P node 10 = M node 11 = NBDD
RESERVED	0-3	Reserved, must be zero (0)

4.4.2. DIRECT_UNIQUE, DIRECT_GROUP, & BROADCAST DATAGRAM



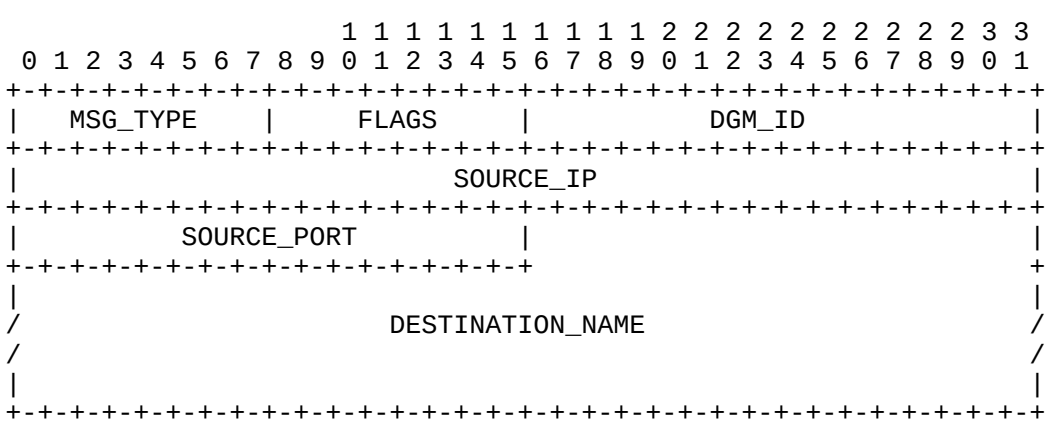
4.4.3. DATAGRAM ERROR PACKET



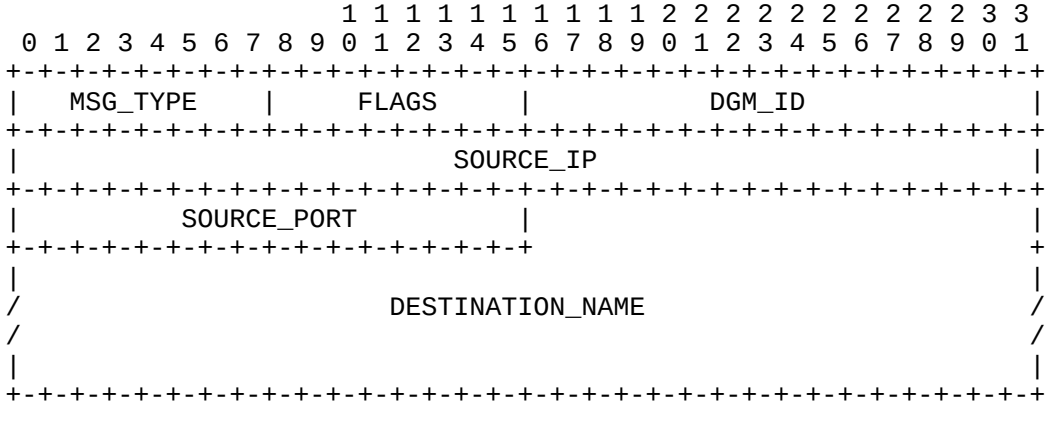
ERROR_CODE values (in hexidecimal):

82 - DESTINATION NAME NOT PRESENT
83 - INVALID SOURCE NAME FORMAT
84 - INVALID DESTINATION NAME FORMAT

4.4.4. DATAGRAM QUERY REQUEST



4.4.5. DATAGRAM POSITIVE AND NEGATIVE QUERY RESPONSE



5. PROTOCOL DESCRIPTIONS

5.1. NAME SERVICE PROTOCOLS

A REQUEST packet is always sent to the well known UDP port - NAME_SERVICE_UDP_PORT. The destination address is normally either the IP broadcast address or the address of the NBNS - the address of the NBNS server it set up at initialization time. In rare cases, a request packet will be sent to an end node, e.g. a NAME QUERY REQUEST sent to "challenge" a node.

A RESPONSE packet is always sent to the source UDP port and source IP address of the request packet.

A DEMAND packet must always be sent to the well known UDP port - NAME_SERVICE_UDP_PORT. There is no restriction on the target IP address.

Terms used in this section:

tid - Transaction ID. This is a value composed from the requestor's IP address and a unique 16 bit value generated by the originator of the transaction.

5.1.1. B-NODE ACTIVITY

5.1.1.1. B-NODE ADD NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a B node
 */
BEGIN

     REPEAT

          /* build name service packet */

          ONT = B_NODE; /* broadcast node */
          G = UNIQUE;   /* unique name */
          TTL = 0;

          broadcast NAME REGISTRATION REQUEST packet;

          /*
           * remote node(s) will send response packet
           * if applicable
           */
```

```
             pause(BCAST_REQ_RETRY_TIMEOUT);

        UNTIL response packet is received or
             retransmit count has been exceeded

        IF no response packet was received THEN
        BEGIN /* no response */
             /*
              * build packet
              */

             ONT = B_NODE; /* broadcast node */
             G = UNIQUE;   /* unique name */
             TTL = 0;

             /*
              * Let other nodes known you have the name
              */

             broadcast NAME UPDATE REQUEST packet;
             /* name can be added to local name table */
             return success;
        END /* no response */
        ELSE
        BEGIN /* got response */

             /*
              * Match return transaction id
              * against tid sent in request
              */

            IF NOT response tid = request tid THEN
            BEGIN
             ignore response packet;
            END
            ELSE
            CASE packet type OF

            NEGATIVE NAME REGISTRATION RESPONSE:

                 return failure; /* name cannot be added */

            POSITIVE NAME REGISTRATION RESPONSE:
            END-NODE CHALLENGE NAME REGISTRATION RESPONSE:

                 /*
                  * B nodes should normally not get this
                  * response.
                  */

                  ignore packet;
```

```
           END /* case */;
        END /* got response */
   END /* procedure */
```

5.1.1.2. B-NODE ADD_GROUP NAME

```
   PROCEDURE add_group_name(newname)

   /*
    * Host initiated processing for a B node
    */

   BEGIN
        /*
         * same as for a unique name with the
         * exception that the group bit (G) must
         * be set in the request packets.
         */

        ...
        G = GROUP;
        ...
        ...

        /*
         * broadcast request ...
         */


   END
```

5.1.1.3. B-NODE FIND_NAME

```
   PROCEDURE find_name(name)

   /*
    * Host initiated processing for a B node
    */

   BEGIN

        REPEAT
             /*
              * build packet
              */
             ONT = B;
             TTL = 0;
             G = DONT CARE;

             broadcast NAME QUERY REQUEST packet;
```

```
             /*
              * a node might send response packet
              */

             pause(BCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
             max transmit threshold exceeded

        IF no response packet received THEN
             return failure;
        ELSE
        IF NOT response tid = request tid THEN
             ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
             /*
              * Start a timer to detect conflict.
              *
              * Be prepared to detect conflict if
              * any more response packets are received.
              *
              */

             save response as authoritative response;
             start_timer(CONFLICT_TIMER);
             return success;

        NEGATIVE NAME QUERY RESPONSE:
        REDIRECT NAME QUERY RESPONSE:

             /*
              * B Node should normally not get either
              * response.
              */

              ignore response packet;

        END /* case */
   END /* procedure */
```

5.1.1.4. B NODE NAME RELEASE

```
   PROCEDURE delete_name (name)
   BEGIN

        REPEAT

             /*
              * build packet
              */
```

```
            ...

            /*
             * send request
             */

            broadcast NAME RELEASE REQUEST packet;

            /*
             * no response packet expected
             */

            pause(BCAST_REQ_RETRY_TIMEOUT);

       UNTIL retransmit count has been exceeded
  END /* procedure */
```

5.1.1.5. B-NODE INCOMING PACKET PROCESSING

Following processing is done when broadcast or unicast packets are received at the NAME_SERVICE_UDP_PORT.

```
  PROCEDURE process_incoming_packet(packet)

  /*
   * Processing initiated by incoming packets for a B node
   */

  BEGIN
       /*
        * Note: response packets are always sent
        * to:
        * source IP address of request packet
        * source UDP port of request packet
        */

       CASE packet type OF

       NAME REGISTRATION REQUEST (UNIQUE):
            IF name exists in local name table THEN
                 send NEGATIVE NAME REGISTRATION RESPONSE ;
       NAME REGISTRATION REQUEST (GROUP):
            IF name exists in local name table THEN
            BEGIN
                 IF local entry is a unique name THEN
                     send NEGATIVE NAME REGISTRATION RESPONSE ;
            END
       NAME QUERY REQUEST:
            IF name exists in local name table THEN
            BEGIN
                 build response packet;
```

```
          send POSITIVE NAME QUERY RESPONSE;
     POSITIVE NAME QUERY RESPONSE:
          IF name conflict timer is not active THEN
          BEGIN
               /*
                * timer has expired already...  ignore this
                * packet
                */

               return;
          END
          ELSE /* timer is active */
          IF a response for this name has previously been
               received THEN
          BEGIN /* existing entry */

               /*
                * we sent out a request packet, and
                * have already received (at least)
                * one response
                *
                * Check if conflict exists.
                * If so, send out a conflict packet.
                *
                * Note: detecting conflict does NOT
                * affect any existing sessions.
                *
                */

               /*
                * Check for name conflict.
                * See "Name Conflict" in Concepts and Methods
                */
               check saved authoritative response against
                    information in this response packet;
               IF conflict detected THEN
               BEGIN
                    unicast NAME CONFLICT DEMAND packet;
                    IF entry exists in cache THEN
                    BEGIN
                         remove entry from cache;
                    END
               END
          END /* existing entry */
          ELSE
          BEGIN
               /*
                * Note: If this was the first response
                * to a name query, it would have been
                * handled in the
                * find_name() procedure.
```

```
                 */

                ignore packet;
           END
      NAME CONFLICT DEMAND:
           IF name exists in local name table THEN
           BEGIN
                mark name as conflict detected;

                /*
                 * a name in the state "conflict detected"
                 * does not "logically" exist on that node.
                 * No further session will be accepted on
                 * that name.
                 * No datagrams can be sent against that name.
                 * Such an entry will not be used for
                 * purposes of processing incoming request
                 * packets.
                 * The only valid user NetBIOS operation
                 * against such a name is DELETE NAME.
                 */
           END
      NAME RELEASE REQUEST:
           IF caching is being done THEN
           BEGIN
                remove entry from cache;
           END
      NAME UPDATE REQUEST:
           IF caching is being done THEN
           BEGIN
                IF entry exists in cache already,
                     update cache;
                ELSE IF name is "interesting" THEN
                BEGIN
                     add entry to cache;
                END
           END

      NODE STATUS REQUEST:
           IF name exists in local name table THEN
           BEGIN
                /*
                 * send only those names that are
                 * in the same scope as the scope
                 * field in the request packet
                 */

                send NODE STATUS RESPONSE;
           END
   END
```

5.1.2. P-NODE ACTIVITY

All packets sent or received by P nodes are unicast UDP packets. A P node sends name service requests to the NBNS node that is specified in the P-node configuration.

5.1.2.1. P-NODE ADD_NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a P node
 */

BEGIN

     REPEAT
          /*
           * build packet
           */

          ONT = P;
          G = UNIQUE;
          ...

          /*
           * send request
           */

          unicast NAME REGISTRATION REQUEST packet;

          /*
           * NBNS will send response packet
           */

          IF receive a WACK RESPONSE THEN
               pause(time from TTL field of response);
          ELSE
               pause(UCAST_REQ_RETRY_TIMEOUT);
     UNTIL response packet is received OR
          retransmit count has been exceeded

     IF no response packet was received THEN
     BEGIN /* no response */
          /*
           * NBNS is down.  Cannot claim name.
           */

          return failure; /* name cannot be claimed */
     END /* no response */
     ELSE
```

```
        BEGIN /* response */
            IF NOT response tid = request tid THEN
            BEGIN
             /*  Packet may belong to another transaction  */
             ignore response packet;
            END
            ELSE
            CASE packet type OF

            POSITIVE NAME REGISTRATION RESPONSE:

                 /*
                  * name can be added
                  */

                 adjust refresh timeout value, TTL, for this name;
                 return success;      /* name can be added */

            NEGATIVE NAME REGISTRATION RESPONSE:
                 return failure; /* name cannot be added */

            END-NODE CHALLENGE REGISTRATION REQUEST:
            BEGIN /* end node challenge */

                 /*
                  * The response packet has in it the
                  * address of the presumed owner of the
                  * name.  Challenge that owner.
                  * If owner either does not
                  * respond or indicates that he no longer
                  * owns the name, claim the name.
                  * Otherwise, the name cannot be claimed.
                  *
                  */

                 REPEAT
                  /*
                   * build packet
                   */
                  ...

                  unicast NAME QUERY REQUEST packet to the
                       address contained in the END NODE
                       CHALLENGE RESPONSE packet;

                  /*
                   * remote node may send response packet
                   */

                  pause(UCAST_REQ_RETRY_TIMEOUT);
```

```
UNTIL response packet is received or
    retransmit count has been exceeded
IF no response packet is received OR
      NEGATIVE NAME QUERY RESPONSE packet
      received THEN
BEGIN /* update */

 /*
  * name can be claimed
  */

 REPEAT

     /*
      * build packet
      */
      ...

     unicast NAME UPDATE REQUEST to NBNS;

     /*
      * NBNS node will send response packet
      */

     IF receive a WACK RESPONSE THEN
           pause(time from TTL field of response);
     ELSE
           pause(UCAST_REQ_RETRY_TIMEOUT);
 UNTIL response packet is received or
     retransmit count has been exceeded
 IF no response packet received THEN
 BEGIN /* no response */

      /*
       * name could not be claimed
       */

      return failure;
 END /* no response */
 ELSE
 CASE packet type OF
      POSITIVE NAME REGISTRATION RESPONSE:
           /*
            * add name
            */
           return success;
      NEGATIVE NAME REGISTRATION RESPONSE:

           /*
            * you lose  ...
            */
```

```
                        return failure;
                    END /* case */
                END /* update */
                ELSE

                /*
                 * received a positive response to the "challenge"
                 * Remote node still has name
                 */

                 return failure;
            END /* end node challenge */
        END /* response */
    END /* procedure */
```

5.1.2.2. P-NODE ADD GROUP NAME

```
    PROCEDURE add_group_name(newname)

    /*
     * Host initiated processing for a P node
     */

    BEGIN
         /*
          * same as for a unique name, except that the
          * request packet must indicate that a
          * group name claim is being made.
          */

         ...
         G = GROUP;
         ...

         /*
          * send packet
          */
          ...


    END
```

5.1.2.3. P-NODE FIND NAME

```
    PROCEDURE find_name(name)

    /*
     * Host initiated processing for a P node
     */

    BEGIN
```

```
        REPEAT
             /*
              * build packet
              */

             ONT = P;
             G = DONT CARE;

             unicast NAME QUERY REQUEST packet;

             /*
              * a NBNS node might send response packet
              */

             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
             max transmit threshold exceeded

        IF no response packet received THEN
             return failure;
        ELSE
        IF NOT response tid = request tid THEN
             ignore packet;
        ELSE
        CASE packet type OF
        POSITIVE NAME QUERY RESPONSE:
             return success;

        REDIRECT NAME QUERY RESPONSE:

             /*
              * NBNS node wants this end node
              * to use some other NBNS node
              * to resolve the query.
              */

              repeat query with NBNS address
                  in the response packet;
        NEGATIVE NAME QUERY RESPONSE:
             return failure;

        END /* case */
   END /* procedure */
```

5.1.2.4. P-NODE DELETE_NAME

```
   PROCEDURE delete_name (name)
```

```
   /*
    * Host initiated processing for a P node
    */

   BEGIN

        REPEAT

             /*
              * build packet
              */
             ...

             /*
              * send request
              */

             unicast NAME RELEASE REQUEST packet;
             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL retransmit count has been exceeded
             or response been received

        IF response has been received THEN
        CASE packet type OF
        POSITIVE NAME RELEASE RESPONSE:
             return success;
        NEGATIVE NAME RELEASE RESPONSE:

             /*
              * NBNS does want node to delete this
              * name !!!
              */

             return failure;
        END /* case */
   END /* procedure */
```

5.1.2.5. P-NODE INCOMING PACKET PROCESSING

Processing initiated by reception of packets at a P node

```
   PROCEDURE process_incoming_packet(packet)

   /*
    * Processing initiated by incoming packets at a P node
    */

   BEGIN
```

```
        /*
         * always ignore UDP broadcast packets
         */

        IF packet was sent as a broadcast THEN
        BEGIN
             ignore packet;
             return;
        END
        CASE packet type of

        NAME CONFLICT DEMAND:
             IF name exists in local name table THEN
                  mark name as in conflict;
             return;

        NAME QUERY REQUEST:
             IF name exists in local name table THEN
             BEGIN /* name exists */

                  /*
                   * build packet
                   */
                  ...

                  /*
                   * send response to the IP address and port
                   * number from which the request was received.
                   */

                  send POSITIVE NAME QUERY RESPONSE ;
                  return;
             END /* exists */
             ELSE
             BEGIN /* does not exist */

                  /*
                   * send response to the requestor
                   */

                  send NEGATIVE NAME QUERY RESPONSE ;
                  return;
             END /* does not exist */
        NODE STATUS REQUEST:
             /*
              * Name of "*" may be used for force node to
              * divulge status for administrative purposes
              */
             IF name in local name table OR name = "*" THEN
             BEGIN
                  /*
```

```
                    * Build response packet and
                    * send to requestor node
                    * Send only those names that are
                    * in the same scope as the scope
                    * in the request packet.
                    */

                    send NODE STATUS RESPONSE;
               END

          NAME RELEASE REQUEST:
               /*
                * This will be received if the NBNS wants to flush the
                * name from the local name table, or from the local
                * cache.
                */

               IF name exists in the local name table THEN
               BEGIN
                    delete name from local name table;
                    inform user that name has been deleted;
               END
               ELSE
                    IF name has been cached locally THEN
                    BEGIN
                         remove entry from cache:
                    END

          END /* case */
     END /* procedure */
```

5.1.2.6. P-NODE TIMER INITIATED PROCESSING

Processing initiated by timer expiration.

```
     PROCEDURE timer_expired()
     /*
      * Processing initiated by the expiration of a timer on a P node
      */
     BEGIN
          /*
           * Send a NAME REFRESH REQUEST for each name which the
           * TTL which has expired.
           */
          REPEAT
               build NAME REFRESH REQUEST packet;
               REPEAT
                    send packet to NBNS;

                    IF receive a WACK RESPONSE THEN
                         pause(time from TTL field of response);
```

```
                ELSE
                     pause(UCAST_REQ_RETRY_TIMEOUT);
           UNTIL response packet is received or
                retransmit count has been exceeded

           CASE packet type OF
             POSITIVE NAME REGISTRATION RESPONSE:
                /* successfully refreshed */
                reset TTL timer for this name;

             NEGATIVE NAME REGISTRATION RESPONSE:
                /*
                 * refused, can't keep name
                 * assume in conflict
                 */
                mark name as in conflict;
           END /* case */

      UNTIL request sent for all names for which TTL
           has expired
END /* procedure */
```

5.1.3. M-NODE ACTIVITY

M nodes behavior is similar to that of P nodes with the addition of some B node-like broadcast actions. M node name service proceeds in two steps:

1. Use broadcast UDP based name service. Depending on the operation, goto step 2.

2. Use directed UDP name service.

The following code for M nodes is exactly the same as for a P node, with the exception that broadcast operations are done before P type operation is attempted.

5.1.3.1. M-NODE ADD NAME

```
PROCEDURE add_name(newname)

/*
 * Host initiated processing for a M node
 */

BEGIN

     /*
      * check if name exists on the
      * broadcast area
      */
```

```
        REPEAT
            /* build packet */

            ....
            broadcast NAME REGISTRATION REQUEST packet;
            pause(BCAST_REQ_RETRY_TIMEOUT);

        UNTIL response packet is received or
             retransmit count has been  exceeded

        IF valid response received THEN
        BEGIN
             /* cannot claim name */

             return failure;
        END

        /*
         * No objections received within the
         * broadcast area.
         * Send request to name server.
         */

        REPEAT
             /*
              * build packet
              */

             ONT = M;
             ...

             unicast NAME REGISTRATION REQUEST packet;

             /*
              * remote NBNS will send response packet
              */

             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);

        UNTIL response packet is received or
             retransmit count has been exceeded

        IF no response packet was received THEN
        BEGIN /* no response */
             /*
              * NBNS is down.  Cannot claim name.
              */
```

```
          return failure; /* name cannot be claimed */
        END /* no response */
        ELSE
        BEGIN /* response */
            IF NOT response tid = request tid THEN
            BEGIN
             ignore response packet;
            END
            ELSE
            CASE packet type OF
            POSITIVE NAME REGISTRATION RESPONSE:

                 /*
                  * name can be added
                  */

                 adjust refresh timeout value, TTL;
                 return success;      /* name can be added */

            NEGATIVE NAME REGISTRATION RESPONSE:
                 return failure; /* name cannot be added */

            END-NODE CHALLENGE REGISTRATION REQUEST:
            BEGIN /* end node challenge */

                 /*
                  * The response packet has in it the
                  * address of the presumed owner of the
                  * name.  Challenge that owner.
                  * If owner either does not
                  * respond or indicates that he no longer
                  * owns the name, claim the name.
                  * Otherwise, the name cannot be claimed.
                  *
                  */

                 REPEAT
                  /*
                   * build packet
                   */
                  ...

                  /*
                   * send packet to address contained in the
                   * response packet
                   */

                  unicast NAME QUERY REQUEST packet;

                  /*
                   * remote node may send response packet
```

```
                 */

                pause(UCAST_REQ_RETRY_TIMEOUT);

               UNTIL response packet is received or
                   retransmit count has been exceeded
               IF no response packet is received THEN
               BEGIN /* no response */

                /*
                 * name can be claimed
                 */
                REPEAT

                    /*
                     * build packet
                     */
                     ...

                    unicast NAME UPDATE REQUEST to NBNS;

                    /*
                     * NBNS node will send response packet
                     */

                    IF receive a WACK RESPONSE THEN
                          pause(time from TTL field of response);
                ELSE
                     pause(UCAST_REQ_RETRY_TIMEOUT);

                UNTIL response packet is received or
                    retransmit count has been exceeded
                IF no response packet received THEN
                BEGIN /* no response */

                     /*
                      * name could not be claimed
                      */

                     return failure;
                END /* no response */
                ELSE
                CASE packet type OF
                POSITIVE NAME REGISTRATION RESPONSE:
                     /*
                      * add name
                      */

                     return success;
                NEGATIVE NAME REGISTRATION RESPONSE:
```

```
                    /*
                     * you lose  ...
                     */

                    return failure;
                END /* case */
              END /* no response */
              ELSE
              IF NOT response tid = request tid THEN
              BEGIN
               ignore response packet;
              END

              /*
               * received a response to the "challenge"
               * packet
               */

              CASE packet type OF
              POSITIVE NAME QUERY:

               /*
                * remote node still has name.
                */

               return failure;
              NEGATIVE NAME QUERY:

               /*
                * remote node no longer has name
                */

               return success;
              END /* case */
          END /* end node challenge */
          END /* case */
      END /* response */
  END /* procedure */
```

5.1.3.2. M-NODE ADD GROUP NAME

```
  PROCEDURE add_group_name(newname)

  /*
   * Host initiated processing for a P node
   */

  BEGIN
      /*
       * same as for a unique name, except that the
       * request packet must indicate that a
```

```
         * group name claim is being made.
         */

        ...
        G = GROUP;
        ...

        /*
         * send packet
         */
         ...


   END
```

5.1.3.3. M-NODE FIND NAME

```
   PROCEDURE find_name(name)

   /*
    * Host initiated processing for a M node
    */

   BEGIN
        /*
         * check if any node on the broadcast
         * area has the name
         */

        REPEAT
             /* build packet */
             ...

             broadcast NAME QUERY REQUEST packet;
             pause(BCAST_REQ_RETRY_TIMEOUT);
        UNTIL response packet received OR
             max transmit threshold exceeded

        IF valid response received THEN
        BEGIN
             save response as authoritative response;
             start_timer(CONFLICT_TIMER);
             return success;
        END

        /*
         * no valid response on the b'cast segment.
         * Try the name server.
         */

        REPEAT
```

```
            /*
             * build packet
             */

            ONT = M;
            G = DONT CARE;

            unicast NAME QUERY REQUEST packet to NBNS;

            /*
             * a NBNS node might send response packet
             */

            IF receive a WACK RESPONSE THEN
                 pause(time from TTL field of response);
            ELSE
                 pause(UCAST_REQ_RETRY_TIMEOUT);
       UNTIL response packet received OR
            max transmit threshold exceeded

       IF no response packet received THEN
            return failure;
       ELSE
       IF NOT response tid = request tid THEN
            ignore packet;
       ELSE
       CASE packet type OF
       POSITIVE NAME QUERY RESPONSE:
            return success;

       REDIRECT NAME QUERY RESPONSE:

            /*
             * NBNS node wants this end node
             * to use some other NBNS node
             * to resolve the query.
             */

             repeat query with NBNS address
                 in the response packet;
       NEGATIVE NAME QUERY RESPONSE:
            return failure;

       END /* case */
   END /* procedure */
```

5.1.3.4. M-NODE DELETE NAME

```
   PROCEDURE delete_name (name)

   /*
```

```
    * Host initiated processing for a P node
    */

   BEGIN
        /*
         * First, delete name on NBNS
         */

        REPEAT

             /*
              * build packet
              */
             ...

             /*
              * send request
              */

             unicast NAME RELEASE REQUEST packet to NBNS;

             IF receive a WACK RESPONSE THEN
                  pause(time from TTL field of response);
             ELSE
                  pause(UCAST_REQ_RETRY_TIMEOUT);
        UNTIL retransmit count has been exceeded
             or response been received

        IF response has been received THEN
        CASE packet type OF
        POSITIVE NAME RELEASE RESPONSE:
             /*
              * Deletion of name on b'cast segment is deferred
              * until after NBNS has deleted the name
              */

             REPEAT
                  /* build packet */

                  ...
                  broadcast NAME RELEASE REQUEST;
                  pause(BCAST_REQ_RETRY_TIMEOUT);
             UNTIL rexmt threshold exceeded

             return success;
        NEGATIVE NAME RELEASE RESPONSE:

             /*
              * NBNS does want node to delete this
              * name
              */
```

```
            return failure;
       END /* case */
   END /* procedure */
```

5.1.3.5. M-NODE INCOMING PACKET PROCESSING

Processing initiated by reception of packets at a M node

```
   PROCEDURE process_incoming_packet(packet)

   /*
    * Processing initiated by incoming packets at a M node
    */

   BEGIN
        CASE packet type of

        NAME CONFLICT DEMAND:
             IF name exists in local name table THEN
                  mark name as in conflict;
             return;

        NAME QUERY REQUEST:
             IF name exists in local name table THEN
             BEGIN /* name exists */

                  /*
                   * build packet
                   */
                  ...

                  /*
                   * send response to the IP address and port
                   * number from which the request was received.
                   */

                  send POSITIVE NAME QUERY RESPONSE ;
                  return;
             END /* exists */
             ELSE
             BEGIN /* does not exist */

                  /*
                   * send response to the requestor
                   */

                  IF request NOT broadcast THEN
                       /*
                        * Don't send negative responses to
                        * queries sent by B nodes
                        */
```

```
                    send NEGATIVE NAME QUERY RESPONSE ;
               return;
          END /* does not exist */
     NODE STATUS REQUEST:
          BEGIN
          /*
           * Name of "*" may be used for force node to
           * divulge status for administrative purposes
           */
          IF name in local name table OR name = "*" THEN
               /*
                * Build response packet and
                * send to requestor node
                * Send only those names that are
                * in the same scope as the scope
                * in the request packet.
                */

               send NODE STATUS RESPONSE;
          END

     NAME RELEASE REQUEST:
          /*
           * This will be received if the NBNS wants to flush the
           * name from the local name table, or from the local
           * cache.
           */

          IF name exists in the local name table THEN
          BEGIN
               delete name from local name table;
               inform user that name has been deleted;
          END
          ELSE
               IF name has been cached locally THEN
               BEGIN
                    remove entry from cache:
               END

     NAME REGISTRATION REQUEST (UNIQUE):
          IF name exists in local name table THEN
               send NEGATIVE NAME REGISTRATION RESPONSE ;
     NAME REGISTRATION REQUEST (GROUP):
          IF name exists in local name table THEN
          BEGIN
               IF local entry is a unique name THEN
                   send NEGATIVE NAME REGISTRATION RESPONSE ;
          END
     END /* case */
END /* procedure */
```

5.1.3.6. M-NODE TIMER INITIATED PROCESSING

Processing initiated by timer expiration:

```
PROCEDURE timer_expired()
/*
 * Processing initiated by the expiration of a timer on a M node
 */
BEGIN
     /*
      * Send a NAME REFRESH REQUEST for each name which the
      * TTL which has expired.
      */
     REPEAT
          build NAME REFRESH REQUEST packet;
          REPEAT
               send packet to NBNS;

               IF receive a WACK RESPONSE THEN
                    pause(time from TTL field of response);
               ELSE
                    pause(UCAST_REQ_RETRY_TIMEOUT);
          UNTIL response packet is received or
               retransmit count has been exceeded

          CASE packet type OF
            POSITIVE NAME REGISTRATION RESPONSE:
               /* successfully refreshed */
               reset TTL timer for this name;

            NEGATIVE NAME REGISTRATION RESPONSE:
               /*
                * refused, can't keep name
                * assume in conflict
                */
               mark name as in conflict;
          END /* case */

     UNTIL request sent for all names for which TTL
          has expired
END /* procedure */
```

5.1.4. NBNS ACTIVITY

A NBNS node will receive directed packets from P and M nodes. Reply packets are always sent as directed packets to the source IP address and UDP port number. Received broadcast packets must be ignored.

5.1.4.1. NBNS INCOMING PACKET PROCESSING

```
   PROCEDURE process_incoming_packet(packet)

   /*
    * Incoming packet processing on a NS node
    */

   BEGIN
        IF packet was sent as a broadcast THEN
        BEGIN
             discard packet;
             return;
        END
        CASE packet type of

        NAME REGISTRATION REQUEST (UNIQUE):
             IF unique name exists in data base THEN
             BEGIN /* unique name exists */
                  /*
                   * NBNS node may be a "passive"
                   * server in that it expects the
                   * end node to do the challenge
                   * server.  Such a NBNS node is
                   * called a "non-secure" server.
                   * A "secure" server will do the
                   * challenging before it sends
                   * back a response packet.
                   */

                  IF non-secure THEN
                  BEGIN
                       /*
                        * build response packet
                        */
                       ...


                       /*
                        * let end node do the challenge
                        */

                       send END-NODE CHALLENGE NAME REGISTRATION
                            RESPONSE;
                       return;
                  END
                  ELSE
                  /*
                   * secure server - do the name
                   * challenge operation
                   */
```

```
REPEAT
    send NAME QUERY REQUEST;
    pause(UCAST_REQ_RETRY_TIMEOUT);
UNTIL response has been received or
     retransmit count has been exceeded
IF no response was received THEN
BEGIN

     /* node down */

     update data base - remove entry;
     update data base - add new entry;
     send POSITIVE NAME REGISTRATION RESPONSE;
     return;
END
ELSE
BEGIN /* challenged node replied */
    /*
     * challenged node replied with
     * a response packet
     */

    CASE packet type

    POSITIVE NAME QUERY RESPONSE:

     /*
      * name still owned by the
      * challenged node
      *
      * build packet and send response
      */
      ...


     /*
      * Note: The NBNS will need to
      * keep track (based on transaction id) of
      * the IP address and port number
      * of the original requestor.
      */

     send NEGATIVE NAME REGISTRATION RESPONSE;
     return;
    NEGATIVE NAME QUERY RESPONSE:

     update data base - remove entry;
     update data base - add new  entry;

     /*
      * build response packet and send
```

```
                    * response
                    */
                   send POSITIVE NAME REGISTRATION RESPONSE;
                   return;
                  END /* case */
             END /* challenged node replied */
        END /* unique name exists in data base */
        ELSE
        IF group name exists in data base THEN
        BEGIN /* group names exists */

             /*
              * Members of a group name are NOT
              * challenged.
              * Make the assumption that
              * at least some of the group members
              * are still alive.
              * Refresh mechanism will
              * allow the NBNS to detect when all
              * members of a group no longer use that
              * name
              */

              send NEGATIVE NAME REGISTRATION RESPONSE;
        END /* group name exists */
        ELSE
        BEGIN /* name does not exist */

             /*
              * Name does not exist in data base
              *
              * This code applies to both non-secure
              * and secure server.
              */

             update data base - add new entry;
             send POSITIVE NAME REGISTRATION RESPONSE;
             return;
        END

   NAME QUERY REQUEST:
        IF name exists in data base THEN
        BEGIN
             /*
              * build response packet and send to
              * requestor
              */
              ...

             send POSITIVE NAME QUERY RESPONSE;
             return;
```

```
            ELSE
            BEGIN
                 /*
                  * build response packet and send to
                  * requestor
                  */
                  ...

                 send NEGATIVE NAME QUERY RESPONSE;
                 return;
            END

       NAME REGISTRATION REQUEST (GROUP):
            IF name exists in data base THEN
            BEGIN
                 IF local entry is a unique name THEN
                 BEGIN /* local is unique */

                     IF non-secure THEN
                     BEGIN
                      send  END-NODE CHALLENGE NAME
                           REGISTRATION RESPONSE;
                      return;
                     END

                     REPEAT
                      send NAME QUERY REQUEST;
                      pause(UCAST_REQ_RETRY_TIMEOUT);
                     UNTIL response received or
                          retransmit count exceeded
                     IF no response received or
                          NEGATIVE NAME QUERY RESPONSE
                           received THEN
                     BEGIN
                      update data base - remove entry;
                      update data base - add new entry;
                      send POSITIVE NAME REGISTRATION RESPONSE;
                      return;
                     END
                     ELSE
                     BEGIN
                      /*
                       * name still being held
                       * by challenged node
                       */

                       send NEGATIVE NAME REGISTRATION RESPONSE;
                     END
                 END /* local is unique */
                 ELSE
                 BEGIN /* local is group  */
```

```
                    /*
                     * existing entry is a group name
                     */

                    update data base - remove entry;
                    update data base - add new entry;
                    send POSITIVE NAME REGISTRATION RESPONSE;
                    return;
               END /* local is group */
          END /* names exists */
          ELSE
          BEGIN /* does not exist */

               /* name does not exist in data base */

               update data base - add new entry;
               send POSITIVE NAME REGISTRATION RESPONSE;
               return;
          END /* does not exist */

     NAME RELEASE REQUEST:

          /*
           * secure server may choose to disallow
           * a node from deleting a name
           */

          update data base - remove entry;
          send POSITIVE NAME RELEASE RESPONSE;
          return;

     NAME UPDATE REQUEST:

          /*
           * End-node completed a successful challenge,
           * no update database
           */

          IF secure server THEN
               send NEGATIVE NAME REGISTRATION RESPONSE;
          ELSE
          BEGIN /* new entry */
               IF entry already exists THEN
                    update data base - remove entry;
               update data base - add new entry;
               send POSITIVE NAME REGISTRATION RESPONSE;
               start_timer(TTL);
          END

     NAME REFRESH REQUEST:
          check for consistency;
```

```
            IF node not allowed to have name THEN
            BEGIN

                 /*
                  * tell end node that it can't have name
                  */
                 send NEGATIVE NAME REGISTRATION RESPONSE;
            END
            ELSE
            BEGIN

                 /*
                  * send confirmation response to the
                  * end node.
                  */
                 send POSITIVE NAME REGISTRATION;
                 start_timer(TTL);
            END
            return;
       END /* case */
  END /* procedure */
```

5.1.4.2. NBNS TIMER INITIATED PROCESSING

A NS node uses timers to flush out entries from the data base. Each entry in the data base is removed when its timer expires. This time value is a multiple of the refresh TTL established when the name was registered.

```
PROCEDURE timer_expired()

/*
 * processing initiated by expiration of TTL for a given name
 */

BEGIN
     /*
      * NBNS can (optionally) ensure
      * that the node is actually down
      * by sending a NODE STATUS REQUEST.
      * If such a request is sent, and
      * no response is received, it can
      * be assumed that the node is down.
      */
     remove entry from data base;
END
```

5.2. SESSION SERVICE PROTOCOLS

The following are variables and should be configurable by the NetBIOS user. The default values of these variables is found in "Defined Constants and Variables" in the Detailed Specification.):

- SSN_RETRY_COUNT - The maximum number TCP connection attempts allowable per a single NetBIOS call request.

- SSN_CLOSE_TIMEOUT is the time period to wait when closing the NetBIOS session before killing the TCP connection if session sends are outstanding.

The following are Defined Constants for the NetBIOS Session Service. (See "Defined Constants and Variables" in the Detailed Specification for the value of these constants):

- SSN_SRVC_TCP_PORT - is the globally well-known TCP port allocated for the NetBIOS Session Service. The service accepts TCP connections on this port to establish NetBIOS Sessions. The TCP connection established to this port by the caller is initially used for the exchange of NetBIOS control information. The actual NetBIOS data connection may also pass through this port or, through the retargetting facility, through another port.

5.2.1. SESSION ESTABLISHMENT PROTOCOLS

5.2.1.1. USER REQUEST PROCESSING

```
PROCEDURE listen(listening name, caller name)
/*
 * User initiated processing for B, P and M nodes
 *
 * This procedure assumes that an incoming session will be
 * retargetted here by a session server.
 */
BEGIN
     Do TCP listen; /* Returns TCP port used */
     Register listen with Session Service, give names and
          TCP port;

     Wait for TCP connection to open;   /* Incoming call */

     Read SESSION REQUEST packet from connection

     Process session request (see section on
          processing initiated by the reception of session
          service packets);
```

```
        Inform Session Service that NetBIOS listen is complete;

        IF session established THEN
             return success and session information to user;
        ELSE
             return failure;
   END /* procedure */

   PROCEDURE call(calling name, called name)
   /*
    * user initiated processing for B, P and M nodes
    */

   /*
    * This algorithm assumes that the called name is a unique name.
    * If the called name is a group name, the call() procedure
    * needs to cycle through the members of the group
    * until either (retry_count == SSN_RETRY_COUNT) or
    * the list has been exhausted.
    */
   BEGIN
        retry_count = 0;
        retarget = FALSE;   /* TRUE: caller is being retargetted */
        name_query = TRUE;  /* TRUE: caller must begin again with */
                            /*       name query. */

        REPEAT
             IF name_query THEN
             BEGIN
                  do name discovery, returns IP address;
                  TCP port = SSN_SRVC_TCP_PORT;

                  IF name discovery fails THEN
                       return failure;
                  ELSE
                       name_query = FALSE;
             END

             /*
              * now have IP address and TCP port of
              * remote party.
              */

             establish TCP connection with remote party, use an
                  ephemeral port as source TCP port;
             IF connection refused THEN
             BEGIN
                  IF retarget THEN
                  BEGIN
                       /* retry */
                       retarget = FALSE;
```

```
                    use original IP address and TCP port;
                    goto LOOP;
               END

               /* retry for just missed TCP listen */

               pause(SESSION_RETRY_TIMER);
               establish TCP connection, again use ephemeral
                    port as source TCP port;

               IF connection refused OR
                  connection timed out THEN
                    return failure;
          END
          ELSE
          IF connection timed out THEN
          BEGIN
               IF retarget THEN
               BEGIN
                    /* retry */
                    retarget = FALSE;
                    use original IP address and TCP port;
                    goto LOOP;
               END
               ELSE
               BEGIN
                    /*
                     * incorrect name discovery was done,
                     * try again
                     */

                    inform name discovery process of
                         possible error;
                    name_query = TRUE;
                    goto LOOP;
               END
          END

          /*
           * TCP connection has been established
           */

          wait for session response packet;
          CASE packet type OF

             POSITIVE SESSION RESPONSE:
               return success and session established
                    information;

             NEGATIVE SESSION RESPONSE:
             BEGIN
```

```
                CASE error OF
                   NOT LISTENING ON CALLED NAME:
                   NOT LISTENING FOR CALLING NAME:
                   BEGIN
                     kill TCP connection;
                     return failure;
                   END

                   CALLED NAME NOT PRESENT:
                   BEGIN
                     /*
                      * called name does not exist on
                      * remote node
                      */

                     inform name discovery procedure
                          of possible error;

                     IF this is a P or M node THEN
                     BEGIN
                          /*
                           * Inform NetBIOS Name Server
                           * it has returned incorrect
                           * information.
                           */
                          send NAME RELEASE REQUEST for called
                            name and IP address to
                            NetBIOS Name Server;
                     END
                     /* retry from beginning */
                     retarget = FALSE;
                     name_query = TRUE;
                     goto LOOP;
                   END /* called name not present */
                END /* case */
             END /* negative response */

             RETARGET SESSION RESPONSE:
             BEGIN
               close TCP connection;
               extract IP address and TCP port from
                    response;
               retarget = TRUE;
             END /* retarget response */
          END /* case */

LOOP:          retry_count = retry_count + 1;

     UNTIL (retry_count > SSN_RETRY_COUNT);
     return failure;
END /* procedure */
```

5.2.1.2. RECEIVED PACKET PROCESSING

These are packets received on a TCP connection before a session has been established. The listen routines attached to a NetBIOS user process need not implement the RETARGET response section. The user process version, separate from a shared Session Service, need only accept (POSITIVE SESSION RESPONSE) or reject (NEGATIVE SESSION RESPONSE) a session request.

```
PROCEDURE session_packet(packet)
/*
 * processing initiated by receipt of a session service
 * packet for a session in the session establishment phase.
 * Assumes the TCP connection has been accepted.
 */
BEGIN
     CASE packet type

        SESSION REQUEST:
        BEGIN
          IF called name does not exist on node THEN
          BEGIN
               send NEGATIVE SESSION RESPONSE with CALLED
                    NAME NOT PRESENT error code;
               close TCP connection;
          END

          Search for a listen with CALLING NAME for CALLED
               NAME;
          IF matching listen is found THEN
          BEGIN
               IF port of listener process is port TCP
                  connection is on THEN
               BEGIN
                    send POSITIVE SESSION RESPONSE;

                    Hand off connection to client process
                         and/or inform user session is
                         established;
               END
               ELSE
               BEGIN
                    send RETARGET SESSION RESPONSE with
                         listener's IP address and
                         TCP port;
                    close TCP connection;
               END
          END
          ELSE
          BEGIN
               /* no matching listen pending */
```

```
                send NEGATIVE SESSION RESPONSE with either
                     NOT LISTENING ON CALLED NAME or NOT
                     LISTENING FOR CALLING NAME error
                     code;
                close TCP connection;
            END
          END /* session request */
        END /* case */
   END /* procedure */
```

5.2.2. SESSION DATA TRANSFER PROTOCOLS

5.2.2.1. USER REQUEST PROCESSING

```
   PROCEDURE send_message(user_message)
   BEGIN
        build SESSION MESSAGE header;
        send SESSION MESSAGE header;
        send user_message;
        reset and restart keep-alive timer;
        IF send fails THEN
        BEGIN
             /*
              * TCP connection has failed */
              */
             close NetBIOS session;
             inform user that session is lost;
             return failure;
        END
        ELSE
             return success;
   END
```

5.2.2.2. RECEIVED PACKET PROCESSING

These are packets received after a session has been established.

```
   PROCEDURE session_packet(packet)
   /*
    * processing initiated by receipt of a session service
    * packet for a session in the data transfer phase.
    */
   BEGIN
        CASE packet type OF

           SESSION  MESSAGE:
           BEGIN
             process message header;
             read in user data;
             reset and restart keep-alive timer;
             deliver data to user;
```

```
           END /* session message */

           SESSION KEEP ALIVE:
             discard packet;

        END /* case */
   END /* procedure */
```

5.2.2.3. PROCESSING INITIATED BY TIMER

```
   PROCEDURE session_ka_timer()
   /*
    * processing initiated when session keep alive timer expires
    */
   BEGIN
            send SESSION KEEP ALIVE, if configured;
            IF send fails THEN
            BEGIN
             /* remote node, or path to it, is down */

             abort TCP connection;
             close NetBIOS session;
             inform user that session is lost;
             return;
            END
   END /* procedure */
```

5.2.3. SESSION TERMINATION PROTOCOLS

5.2.3.1. USER REQUEST PROCESSING

```
   PROCEDURE close_session()

   /* initiated by a user request to close a session */

   BEGIN
        close gracefully the TCP connection;

        WAIT for the connection to close or SSN_CLOSE_TIMEOUT
             to expire;

        IF time out expired THEN
             abort TCP connection;
   END /* procedure */
```

5.2.3.2. RECEPTION INDICATION PROCESSING

```
   PROCEDURE close_indication()
   /*
    * initiated by a TCP indication of a close request from
    * the remote connection partner.
```

```
    */
   BEGIN
        close gracefully TCP connection;

        close NetBIOS session;

        inform user session closed by remote partner;
   END /* procedure */
```

5.3. NetBIOS DATAGRAM SERVICE PROTOCOLS

The following are GLOBAL variables and should be NetBIOS user configurable:

- SCOPE_ID: the non-leaf section of the domain name preceded by a '.' which represents the domain of the NetBIOS scope for the NetBIOS name. The following protocol description only supports single scope operation.

- MAX_DATAGRAM_LENGTH: the maximum length of an IP datagram. The minimal maximum length defined in for IP is 576 bytes. This value is used when determining whether to fragment a NetBIOS datagram. Implementations are expected to be capable of receiving unfragmented NetBIOS datagrams up to their maximum size.

- BROADCAST_ADDRESS: the IP address B-nodes use to send datagrams with group name destinations and broadcast datagrams. The default is the IP broadcast address for a single IP network.

The following are Defined Constants for the NetBIOS Datagram Service:

- DGM_SRVC_UDP_PORT: the globally well-known UDP port allocated where the NetBIOS Datagram Service receives UDP packets. See section 6, "Defined Constants", for its value.

5.3.1. B NODE TRANSMISSION OF NetBIOS DATAGRAMS

```
   PROCEDURE send_datagram(data, source, destination, broadcast)

   /*
    * user initiated processing on B node
    */

   BEGIN
        group = FALSE;

        do name discovery on destination name, returns name type and
             IP address;
```

```
        IF name type is group name THEN
        BEGIN
             group = TRUE;
        END

        /*
         * build datagram service UDP packet;
         */
        convert source and destination NetBIOS names into
             half-ASCII, biased encoded name;
        SOURCE_NAME = cat(source, SCOPE_ID);
        SOURCE_IP = this nodes IP address;
        SOURCE_PORT =  DGM_SRVC_UDP_PORT;

        IF NetBIOS broadcast THEN
        BEGIN
             DESTINATION_NAME = cat("*", SCOPE_ID)
        END
        ELSE
        BEGIN
             DESTINATION_NAME = cat(destination, SCOPE_ID)
        END

        MSG_TYPE = select_one_from_set
             {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
        DGM_ID = next transaction id for Datagrams;
        DGM_LENGTH = length of data + length of second level encoded
             source and destination names;

        IF (length of the NetBIOS Datagram, including UDP and
            IP headers, > MAX_DATAGRAM_LENGTH) THEN
        BEGIN
             /*
              * fragment NetBIOS datagram into 2 UDP packets
              */
             Put names into 1st UDP packet and any data that fits
                  after names;
             Set MORE and FIRST bits in 1st UDP packet's FLAGS;
             OFFSET in 1st UDP = 0;

             Replicate NetBIOS Datagram header from 1st UDP packet
                  into 2nd UDP packet;
             Put rest of data in 2nd UDP packet;
             Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
             OFFSET in 2nd UDP = DGM_LENGTH - number of name and
                  data bytes in 1st UDP;
        END
        BEGIN
             /*
              * Only need one UDP packet
              */
```

```
             USER_DATA = data;
             Clear MORE bit and set FIRST bit in FLAGS;
             OFFSET = 0;
        END

        IF (group == TRUE) OR (NetBIOS broadcast) THEN
        BEGIN
             send UDP packet(s) to BROADCAST_ADDRESS;
        END
        ELSE
        BEGIN
             send UDP packet(s) to IP address returned by name
                discovery;
        END
   END /* procedure */
```

5.3.2. P AND M NODE TRANSMISSION OF NetBIOS DATAGRAMS

```
   PROCEDURE send_datagram(data, source, destination, broadcast)

   /*
    * User initiated processing on P and M node.
    *
    * This processing is the same as for B nodes except for
    * sending broadcast and multicast NetBIOS datagrams.
    */

   BEGIN
        group = FALSE;

        do name discovery on destination name, returns name type
             and IP address;
        IF name type is group name THEN
        BEGIN
             group = TRUE;
        END

        /*
         * build datagram service UDP packet;
         */
        convert source and destination NetBIOS names into
             half-ASCII, biased encoded name;
        SOURCE_NAME = cat(source, SCOPE_ID);
        SOURCE_IP = this nodes IP address;
        SOURCE_PORT =  DGM_SRVC_UDP_PORT;

        IF NetBIOS broadcast THEN
        BEGIN
             DESTINATION_NAME = cat("*", SCOPE_ID)
        END
        ELSE
```

```
        BEGIN
             DESTINATION_NAME = cat(destination, SCOPE_ID)
        END

        MSG_TYPE = select_one_from_set
             {BROADCAST, DIRECT_UNIQUE, DIRECT_GROUP}
        DGM_ID = next transaction id for Datagrams;
        DGM_LENGTH = length of data + length of second level encoded
             source and destination names;

        IF (length of the NetBIOS Datagram, including UDP and
            IP headers, > MAX_DATAGRAM_LENGTH) THEN
        BEGIN
             /*
              * fragment NetBIOS datagram into 2 UDP packets
              */
             Put names into 1st UDP packet and any data that fits
                  after names;
             Set MORE and FIRST bits in 1st UDP packet's FLAGS;

             OFFSET in 1st UDP = 0;

             Replicate NetBIOS Datagram header from 1st UDP packet
                  into 2nd UDP packet;
             Put rest of data in 2nd UDP packet;
             Clear MORE and FIRST bits in 2nd UDP packet's FLAGS;
             OFFSET in 2nd UDP = DGM_LENGTH - number of name and
                  data bytes in 1st UDP;
        END
        BEGIN
             /*
              * Only need one UDP packet
              */
             USER_DATA = data;
             Clear MORE bit and set FIRST bit in FLAGS;
             OFFSET = 0;
        END

        IF (group == TRUE) OR (NetBIOS broadcast) THEN
        BEGIN
             /*
              * Sending of following query is optional.
              * Node may send datagram to NBDD immediately
              * but NBDD may discard the datagram.
              */
             send DATAGRAM QUERY REQUEST to NBDD;
             IF response is POSITIVE QUERY RESPONSE THEN
                  send UDP packet(s) to NBDD Server IP address;
             ELSE
             BEGIN
                  get list of destination nodes from NBNS;
```

```
                FOR EACH node in list
                BEGIN
                     send UDP packet(s) to this node's
                          IP address;
                END
           END
      END
      ELSE
      BEGIN
           send UDP packet(s) to IP address returned by name
              discovery;
END /* procedure */
```

5.3.3. RECEPTION OF NetBIOS DATAGRAMS BY ALL NODES

The following algorithm discards out of order NetBIOS Datagram fragments. An implementation which reassembles out of order NetBIOS Datagram fragments conforms to this specification. The fragment discard timer is initialized to the value FRAGMENT_TO. This value should be user configurable. The default value is given in Section 6, "Defined Constants and Variables".

```
PROCEDURE datagram_packet(packet)

/*
 * processing initiated by datagram packet reception
 * on B, P and M nodes
 */
BEGIN
     /*
      * if this node is a P node, ignore
      * broadcast packets.
      */

     IF this is a P node AND incoming packet is
          a broadcast packet THEN
     BEGIN
          discard packet;
     END

     CASE packet type OF

        DATAGRAM SERVICE:
        BEGIN
          IF FIRST bit in FLAGS is set THEN
          BEGIN
               IF MORE bit in FLAGS is set THEN
               BEGIN
                    Save 1st UDP packet of the Datagram;
                    Set this Datagram's fragment discard
                      timer to FRAGMENT_TO;
```

```
                    return;
               END
               ELSE
                    Datagram is composed of a single
                      UDP packet;
          END
          ELSE
          BEGIN
               /* Have the second fragment of a Datagram */

               Search for 1st fragment by source IP address
                  and DGM_ID;
               IF found 1st fragment THEN
                    Process both UDP packets;
               ELSE
               BEGIN
                    discard 2nd fragment UDP packet;
                    return;
               END
          END

          IF DESTINATION_NAME is '*' THEN
          BEGIN
               /* NetBIOS broadcast */

               deliver USER_DATA from UDP packet(s) to all
                    outstanding receive broadcast
                    datagram requests;
               return;
          END
          ELSE
          BEGIN /* non-broadcast */
               /* Datagram for Unique or Group Name */

               IF DESTINATION_NAME is not present in the
                  local name table THEN
               BEGIN
                    /* destination not present */
                    build DATAGRAM ERROR packet, clear
                         FIRST and MORE bit, put in
                         this nodes IP and PORT, set
                         ERROR_CODE;
                    send DATAGRAM ERROR packet to
                         source IP address and port
                         of UDP;
                    discard UDP packet(s);
                    return;
               END
               ELSE
               BEGIN /* good */
                    /*
```

```
                    * Replicate received NetBIOS datagram for
                    * each recipient
                    */
                   FOR EACH pending NetBIOS user's receive
                        datagram operation
                   BEGIN
                        IF source name of operation
                           matches destination name
                           of packet THEN
                        BEGIN
                           deliver USER_DATA from UDP
                             packet(s);
                        END
                   END /* for each */
                   return;
              END /* good */
         END /* non-broadcast */
        END /* datagram service */

      DATAGRAM ERROR:
      BEGIN
           /*
            * name service returned incorrect information
            */

           inform local name service that incorrect
             information was provided;

           IF this is a P or M node THEN
           BEGIN
                /*
                 * tell NetBIOS Name Server that it may
                 * have given incorrect information
                 */

                send NAME RELEASE REQUEST with name
                  and incorrect IP address to NetBIOS
                  Name Server;
           END
      END /* datagram error */

   END /* case */
END
```

5.3.4. PROTOCOLS FOR THE NBDD

The key to NetBIOS Datagram forwarding service is the packet delivered to the destination end node must have the same NetBIOS header as if the source end node sent the packet directly to the destination end node. Consequently, the NBDD does not reassemble NetBIOS Datagrams. It forwards the UDP packet as is.

```
   PROCEDURE  datagram_packet(packet)

   /*
    * processing initiated by a incoming datagram service
    * packet on a NBDD node.
    */

   BEGIN
        CASE packet type OF

           DATAGRAM SERVICE:
           BEGIN
                IF packet was sent as a directed
                   NetBIOS datagram THEN
                BEGIN
                  /*
                   * provide group forwarding service
                   *
                   * Forward datagram to each member of the
                   * group.  Can forward via:
                   *   1) get list of group members and send
                   *   the DATAGRAM SERVICE packet unicast
                   *   to each
                   *   2) use Group Multicast, if available
                   *   3) combination of 1) and 2)
                   */

                  ...

                END

                ELSE
                BEGIN
                  /*
                   * provide broadcast forwarding service
                   *
                   * Forward datagram to every node in the
                   * NetBIOS scope.  Can forward via:
                   *   1) get list of group members and send
                   *   the DATAGRAM SERVICE packet unicast
                   *   to each
                   *   2) use Group Multicast, if available
                   *   3) combination of 1) and 2)
                   */

                  ...

                END
           END /* datagram service */

           DATAGRAM ERROR:
```

```
          BEGIN
            /*
             * Should never receive these because Datagrams
             * forwarded have source end node IP address and
             * port in NetBIOS header.
             */

            send DELETE NAME REQUEST with incorrect name and
                 IP address to NetBIOS Name Server;

          END /* datagram error */

          DATAGRAM QUERY REQUEST:
          BEGIN
            IF can send packet to DESTINATION_NAME THEN
            BEGIN
                 /*
                  * NBDD is able to relay Datagrams for
                  * this name
                  */

                 send POSITIVE DATAGRAM QUERY RESPONSE to
                   REQUEST source IP address and UDP port
                   with request's DGM_ID;
            END
            ELSE
            BEGIN
                 /*
                  * NBDD is NOT able to relay Datagrams for
                  * this name
                  */

                 send NEGATIVE DATAGRAM QUERY RESPONSE to
                   REQUEST source IP address and UDP port

                   with request's DGM_ID;
            END
          END /* datagram query request */

       END /* case */
    END /* procedure */
```

6. DEFINED CONSTANTS AND VARIABLES

GENERAL:

SCOPE_ID	The name of the NetBIOS scope. This is expressed as a character string meeting the requirements of the domain name system and without a leading or trailing "dot". An implementation may elect to make this a single global value for the node or allow it to be specified with each separate NetBIOS name (thus permitting cross-scope references.)
BROADCAST_ADDRESS	An IP address composed of the nodes's network and subnetwork numbers with all remaining bits set to one. I.e. "Specific subnet" broadcast addressing according to section 2.3 of RFC 950.
BCAST_REQ_RETRY_TIMEOUT	250 milliseconds. An adaptive timer may be used.
BCAST_REQ_RETRY_COUNT	3
UCAST_REQ_RETRY_TIMEOUT	5 seconds An adaptive timer may be used.
UCAST_REQ_RETRY_COUNT	3
MAX_DATAGRAM_LENGTH	576 bytes (default)

NAME SERVICE:

REFRESH_TIMER	Negotiated with NBNS for each name.
CONFLICT_TIMER	1 second Implementations may chose a longer value.
NAME_SERVICE_TCP_PORT	137 (decimal)

NAME_SERVICE_UDP_PORT	137 (decimal)
INFINITE_TTL	0

SESSION SERVICE:

SSN_SRVC_TCP_PORT	139 (decimal)
SSN_RETRY_COUNT	4 (default) Re-configurable by user.
SSN_CLOSE_TIMEOUT	30 seconds (default) Re-configurable by user.
SSN_KEEP_ALIVE_TIMEOUT	60 seconds, recommended, may be set to a higher value. (Session keep-alives are used only if configured.)

DATAGRAM SERVICE:

DGM_SRVC_UDP_PORT	138 (decimal)
FRAGMENT_TO	2 seconds (default)

REFERENCES

[1] "Protocol Standard For a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987.

[2] J. Reynolds, J. Postel, "Assigned Numbers", RFC 990, November 1986.

[3] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, November 1983.